A Dense Hierarchy of Sublinear Time Approximation Schemes for Bin Packing

Richard Beigel[1] and Bin Fu[2]

[1]CIS Department
Temple University, Philadelphia PA 19122-6094, USA
beigel@cis.temple.edu

[2]Department of Computer Science
University of Texas-Pan American, Edinburg, TX 78539, USA
binfu@cs.panam.edu

Abstract

The bin packing problem is to find the minimum number of bins of size one to pack a list of items with sizes $a_1, \ldots, a_n$ in $(0, 1]$. Using uniform sampling, which selects a random element from the input list each time, we develop a randomized $O(\frac{n(\log n)(\log\log n)}{\sum_{i=1}^{n} a_i} + (\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})})$ time $(1+\epsilon)$-approximation scheme for the bin packing problem. We show that every randomized algorithm with uniform random sampling needs $\Omega(\frac{n}{\sum_{i=1}^{n} a_i})$ time to give an $(1+\epsilon)$-approximation. For each function $s(n) : N \to N$, define $\sum(s(n))$ to be the set of all bin packing problems with the sum of item sizes equal to $s(n)$. For a constant $b \in (0, 1)$, every problem in $\sum(n^b)$ has an $O(n^{1-b}(\log n)(\log\log n) + (\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})})$ time $(1+\epsilon)$-approximation for an arbitrary constant ϵ. On the other hand, there is no $o(n^{1-b})$ time $(1+\epsilon)$-approximation scheme for the bin packing problems in $\sum(n^b)$ for some constant $\epsilon > 0$. We show that $\sum(n^b)$ is NP-hard for every $b \in (0, 1]$. This implies a dense sublinear time hierarchy of approximation schemes for a class of NP-hard problems, which are derived from the bin packing problem. We also show a randomized streaming approximation scheme for the bin packing problem such that it needs only constant updating time and constant space, and outputs an $(1+\epsilon)$-approximation in $(\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})}$ time. Let $S(\delta)$-bin packing be the class of bin packing problems with each input item of size at least δ. This research also gives a natural example of NP-hard problem ($S(\delta)$-bin packing) that has a constant time approximation scheme, and a constant time and space sliding window streaming approximation scheme, where δ is a positive constant.

1. Introduction

The bin packing problem is to find the minimum number of bins of size one to pack a list of items with sizes $a_1, \ldots, a_n$ in $(0, 1]$. It is a classical NP-hard problem and has been widely studied. The bin packing problem has many applications in the engineering and information sciences. Some approximation algorithm has been developed for bin packing problem: for examples, the first fit, best fit, sum-of-squares, or Gilmore-Gomory cuts [2, 8, 7, 16, 15]. The first linear time approximation scheme is shown in [11]. Recently, a sublinear time $O(\sqrt{n})$ with weighted sampling and a sublinear time $O(n^{1/3})$ with a combination of weighted and uniform samplings were shown for bin packing problem [3].

We study the bin packing problem in randomized offline sublinear time model, randomized streaming model, and randomized sliding window streaming model. We also study the bin packing problem that has input item sizes to be random numbers in $[0, 1]$. Sublinear time algorithms have been found for many computational problems, such as checking polygon intersections [5], estimating the cost of a minimum spanning tree [6, 9, 10], finding geometric separators [13], and property testing [22, 17], etc. Early research on streaming algorithms dealt with simple statistics of the input data streams, such as the median [21], the number of distinct elements [12], or frequency moments [1]. Streaming algorithm is becoming more and more important due to the development of internet, which brings a lot of applications. There are many streaming algorithms that have been proposed from the areas of computational theory, database, and networking, etc.

Due to the important role of bin packing problem in the development of algorithm design and its application in many other fields, it is essential to study the bin packing problem in these natural models. Our offline approximation scheme is based on the uniform sampling, which selects a random element from the input list each time. Our first approach is to approximate the bin packing problem with a small number of samples under uniform sampling. We identify that the complexity of approximation for the bin packing problem inversely depends on the sum of the sizes of input items.

Using uniform sampling, we develop a randomized $O(\frac{n(\log n)(\log\log n)}{\sum_{i=1}^{n} a_i} + (\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})})$ time $(1+\epsilon)$-approximation scheme for the bin packing problem. We show that every randomized algorithm with uniform random sampling needs $\Omega(\frac{n}{\sum_{i=1}^{n} a_i})$ time to give an $(1+\epsilon)$-approximation. Based on an adaptive random sampling method developed in this paper, our algorithm automatically detects an approximation to the weights of summation of the input items in time $O(\frac{n(\log n)(\log\log n)}{\sum_{i=1}^{n} a_i})$ time, and then yields an $(1+\epsilon)$-approximation.

For each function $s(n) : N \to N$, define $\sum(s(n))$ to be the set of all bin packing problems with the sum of item sizes equal to $s(n)$. For a constant $b \in (0, 1)$, every problem in $\sum(n^b)$ has an $O(n^{1-b}(\log n)(\log\log n) + (\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})})$ time $(1+\epsilon)$-approximation for an arbitrary constant ϵ. On the other hand, there is no $o(n^{1-b})$ time $(1+\epsilon)$-approximation scheme for the bin packing problems in $\sum(n^b)$ for some constant $\epsilon > 0$. We show that $\sum(n^b)$ is NP-hard for every $b \in (0, 1]$. This implies a dense sublinear time hierarchy of approximation schemes for a class of NP-hard problems that are derived from bin packing problem. We also show a randomized single pass streaming approximation scheme for the bin packing problem such that it needs only constant updating time and constant space, and outputs an $(1+\epsilon)$-approximation in $(\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})}$ time. This research also gives an natural example of NP-hard problem that has a constant time approximation scheme, and a constant time and space sliding window single pass streaming approximation scheme.

The streaming algorithms in this paper for bin packing problem only approximate the minimum number of bins to pack those input items. It also gives a packing plan that allows an item position to be changed at different moment. This has no contradiction with the existing lower bound [4, 19] that no approximation scheme exists for online algorithm that does not change bins of already packed items.

A more general model of bin packing is studied in this paper. Given a list of items in $(0, 1]$, allocate them to several kinds of bins with variant sizes and weights. We want to minimize the total

costs $\sum_{i=1}^{k} u_i w_i$, where u_i is the number of bins of size s_i and cost w_i.

In section 2, we give a description of computational models used in this paper. A brief description of our methods are also presented. In section 3, we show an adaptive random sampling method for the bin packing problem. In section 6, we present randomized algorithms and their lower bound for offline bin packing problem. In section 8, we show a streaming approximation scheme for bin packing problem. In section 9, we show a sliding window streaming approximation scheme for bin packing problem with each input item of size at least a positive constant δ. The main result of this paper is stated in Theorem 10.

2. Models of Computation and Overview of Methods

Algorithms for bin packing problem in this paper are under four models, which are deterministic, randomized, streaming, and sliding windows streaming models.

Definition 1.

- A *bin packing* is an allocation of the input items of sizes $a_1, \ldots, a_n$ in $(0, 1]$ to bins of size 1. We want to minimize the total number of bins. We often use $Opt(L)$ to denote the least number bins for packing items in L.
- Assume that c and η are constants in $(0, 1)$, and k is a constant integer. There are k kinds of bins of different sizes. If $c \leq s_i \leq 1$, and $\eta \leq w_i \leq 1$ for all $i = 1, 2, \ldots, k$, then we call the k kinds of bins to be (c, η, k)-*related*, where w_i and s_i are the cost and size of the i-th kind of bin, respectively.
- A *bin packing with* (c, η, k)-*related bins* is to allocate the input items $a_1, \ldots, a_n$ in $(0, 1]$ to (c, η, k)-related bins. We want to minimize the total costs $\sum_{i=1}^{k} u_i w_i$, where u_i is the number of bins of cost w_i. We often use $Opt_{c,\eta,k}(L)$ to denote the least cost for packing items in L with (c, η, k)-related bins. It is easy to see $Opt(L) = Opt_{1,1,1}(L)$.
- For a positive constant δ, a $S(\delta)$-bin packing problem is the bin packing problem with all input items at least δ.
- For a nondecreasing function $f(n) : N \to N$, a $\sum(f(n))$-bin packing problem is the bin packing problem with all input items $a_1, \ldots, a_n$ satisfying $\sum_{i=1}^{n} a_i = f(n)$.

Deterministic Model: The bin packing problem under the deterministic model has been well studied. We give a generalized version of bin packing problem that allows multiple sizes of bins to pack them. It is called as bin packing with (c, η, k) related bins in Definition 1. It is presented in Section 5.

Randomized Models: Our main model of computation is based on the uniform random sampling. We give the definitions for both uniform and weighted random samplings below.

Definition 2. Assume that $a_1, \ldots, a_n$ is an input list of items in $(0, 1]$ for a bin packing problem.

- A *uniform sampling* selects an element a from the input list with $\Pr[a = a_i] = \frac{1}{n}$ for $i = 1, \ldots, n$.
- A *weighted sampling* selects an element a from the input list with $\Pr[a = a_i] = \frac{a_i}{\sum_{i=1}^{n} a_i}$ for $i = 1, \ldots, n$.

We feel that the uniform sampling is more practical to implement than weighted sampling. In this paper, our offline randomized algorithms are based on uniform sampling. The weighted sampling was used in [3]. The description of our offline algorithm with uniform random sampling is given in Section 6.

Streaming Computation: A data stream is an ordered sequence of data items $p_1, p_2, \ldots, p_n$. Here, n denotes the number of data points in the stream. A *streaming algorithm* is an algorithm

that computes some function over a data stream and has the following properties: 1. The input data are accessed in the sequential order of the data stream. 2. The order of the data items in the stream is not controlled by the algorithm. Our algorithm for this model is presented in Section 8.

Sliding Window Model: In the sliding window streaming model, there is a window size n for the most recent n items. The bin packing problem for the sliding window streaming algorithm is to pack the most recent n items. Our algorithm for this model is presented in Section 9.

Bin Packing with Random Inputs: We study the bin packing problem such that the input is a series of sizes that are random numbers in $[0, 1]$. It has a constant time approximation scheme and will be presented in Section 9.1.

2.1. Overview of Our Method

We develop algorithms for the bin packing problem under offline uniform random sampling model, the streaming computation model, and sliding window streaming model (only for $S(\delta)$-bin packing with a positive constant δ). The brief ideas are given below.

2.1.1. Sublinear Time Algorithm for Offline Bin Packing

Since the sum of input item sizes is not a part of input, it needs $O(n)$ time to compute its exact value, and it's unlikely to be approximated via one round random sampling in a sublinear time. We first approximate the sum of sizes of items through a multi-phase adaptive random sampling. Select a constant φ to be the threshold for large items. Select a small constant $\gamma = O(\epsilon)$. All the items from the input are partitioned into intervals $[\pi_1, \pi_0], (\pi_2, \pi_1] \ldots, (\pi_{i+1}, \pi_i], \ldots$ such that $\pi_0 = 1, \pi_1 = \varphi$, and $\pi_{i+1} = \pi_i/(1+\gamma)$ for $i = 2, \ldots$. We approximate the number of items in each interval $(\pi_{i+1}, \pi_i]$ via uniform random sampling. Those intervals with very a small number of items will be dropped. This does not affect much of the ratio of approximation. One of worst cases is that all small items are of size $\frac{1}{n^2}$ and all large size items are of size 1. In this case, we need to sample $\Omega(\frac{n}{\sum_{a_i=1} 1})$ number of items to approximate the number of 1s. This makes the total time to be $\Omega(\frac{n}{\sum_{i=1}^{n} a_i})$. Packing the items of large size is adapted the method in [11], which uses a linear programming method to pack the set of all large items, and fills small items into those bins with large items to waste only a small piece of space for each bin. Then the small items are put into bins that still have space left after packing large items. When the sum of all item sizes is $O(1)$, we need $O(n)$ time. Thus, the $O(n)$ time algorithm is a part of our algorithm for the case $\sum_{i=1}^{n} a_i = O(1)$.

2.1.2. Streaming Algorithm for Bin Packing

We apply the above approximation scheme to construct a single pass streaming algorithm for bin packing problem. A crucial step is to sample some random elements among those input items of size at least δ, which is set according to ϵ. The weights of small items are added to a variable s_1. After packing large items of size at least δ, we pack small items into those bins so that each bin does not waste more than δ space while there is small items unpacked.

2.1.3. Sliding Window Streaming Algorithm for $S(\delta)$-Bin Packing

Our sliding window single pass streaming algorithm deals with the bin packing problem that all input items are of size at least a constant δ. Let n be the size of sliding window instead of the total number of input items. Select a sufficiently large constant k. There are k sessions to approximate the bin packing. After receiving every $\frac{n}{k}$ items, a new session is started to approximate the bin packing. The approximation ratio is guaranteed via ignoring at most $\frac{n}{k}$ items. As each item is of large size at least δ, ignoring $\frac{n}{k}$ items only affect a small ratio of approximation.

2.1.4. Chernoff Bounds

The analysis of our randomized algorithm often use the well known Chernoff bounds, which are described below. All proofs of this paper are self-contained except the following famous theorems in probability theory and the existence of a polynomial time algorithm for linear programming.

Theorem 3 ([20]). *Let $X_1, \ldots, X_n$ be n independent random 0-1 variables, where X_i takes 1 with probability p_i. Let $X = \sum_{i=1}^{n} X_i$, and $\mu = E[X]$. Then for any $\delta > 0$,*

i. $\Pr(X < (1-\delta)\mu) < e^{-\frac{1}{2}\mu\delta^2}$, *and*

ii. $\Pr(X > (1+\delta)\mu) < \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{\mu}$.

We follow the proof of Theorem 3 to make the following versions (Theorem 5, Theorem 4, and Corollary 6) of Chernoff bound for our algorithm analysis.

Theorem 4. *Let $X_1, \ldots, X_n$ be n independent random 0-1 variables, where X_i takes 1 with probability at least p for $i = 1, \ldots, n$. Let $X = \sum_{i=1}^{n} X_i$, and $\mu = E[X]$. Then for any $\delta > 0$,* $\Pr(X < (1-\delta)pn) < e^{-\frac{1}{2}\delta^2 pn}$.

Theorem 5. *Let $X_1, \ldots, X_n$ be n independent random 0-1 variables, where X_i takes 1 with probability at most p for $i = 1, \ldots, n$. Let $X = \sum_{i=1}^{n} X_i$. Then for any $\delta > 0$,* $\Pr(X > (1+\delta)pn) < \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{pn}$.

Define $g_1(\delta) = e^{-\frac{1}{2}\delta^2}$ and $g_2(\delta) = \frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}$. Define $g(\delta) = \max(g_1(\delta), g_2(\delta))$. We note that $g_1(\delta)$ and $g_2(\delta)$ are always strictly less than 1 for all $\delta > 0$. It is trivial for $g_1(\delta)$. For $g_2(\delta)$, this can be verified by checking that the function $f(x) = (1+x)\ln(1+x) - x$ is increasing and $f(0) = 0$. This is because $f'(x) = \ln(1+x)$ which is strictly greater than 0 for all $x > 0$.

Corollary 6 ([18]). *Let $X_1, \ldots, X_n$ be n independent random 0-1 variables and $X = \sum_{i=1}^{n} X_i$.*

i. If X_i takes 1 with probability at most p for $i = 1, \ldots, n$, then for any $\frac{1}{3} > \epsilon > 0$, $\Pr(X > pn + \epsilon n) < e^{-\frac{1}{3}n\epsilon^2}$.

ii. If X_i takes 1 with probability at least p for $i = 1, \ldots, n$, then for any $\epsilon > 0$, $\Pr(X < pn - \epsilon n) < e^{-\frac{1}{2}n\epsilon^2}$.

A well known fact in probability theory is the inequality

$$\Pr(E_1 \cup E_2 \ldots \cup E_m) \leq \Pr(E_1) + \Pr(E_2) + \ldots + \Pr(E_m),$$

where $E_1, E_2, \ldots, E_m$ are m events that may not be independent. In the analysis of our randomized algorithm, there are multiple events such that the failure from any of them may fail the entire algorithm. We often characterize the failure probability of each of those events, and use the above inequality to show that the whole algorithm has a small chance to fail after showing that each of them has a small chance to fail.

3. Adaptive Random Sampling for Bin Packing

In this section, we develop an adaptive random sampling method to get the rough information for a list of items for the bin packing problem. We show a randomized algorithm to approximate the sum of the sizes of input items in $O((\frac{n}{\sum_{i=1}^{n} a_i})(\log n)\log\log n))$ time. This is the core step of our randomized algorithm, and is also or main technical contribution.

Definition 7.

- For each interval I and a list of items S, define $C(I, S)$ to be the number of items of S in I.
- For φ, δ, and γ in $(0, 1)$, a $(\varphi, \delta, \gamma)$-*partition* for $(0, 1]$ divides the interval $(0, 1]$ into intervals $I_1 = [\pi_1, \pi_0], I_2 = (\pi_2, \pi_1], I_3 = (\pi_3, \pi_2], \ldots, I_k = (0, \pi_{k-1}]$ such that $\pi_0 = 1, \pi_1 = \varphi, \pi_i = \pi_{i-1}(1 - \delta)$ for $i = 2, \ldots, k - 1$, and π_{k-1} is the first element $\pi_{k-1} \leq \frac{\gamma}{n^2}$.
- For a set A, $|A|$ is the number of elements in A. For a list S of items, $|S|$ is the number of items in S.

Lemma 8. *For parameters φ, δ, and γ in $(0, 1)$, a $(\varphi, \delta, \gamma)$-partition for $(0, 1]$ has the number of intervals $k \leq \frac{2 \log n}{\gamma \theta}$.*

Proof: The number of intervals k is the least integer with $\delta(1 - \delta)^k \leq (1 - \delta)^k \leq \frac{\gamma}{n^2}$. We have $k \leq \frac{\log \frac{n^2}{\gamma}}{\log(1-\delta)} \leq \frac{2 \log n}{\gamma \delta}$. ∎

We need to approximate the number of large items, the total sum of the sizes of items, and the total sum of the sizes of small items. For a $(\varphi, \delta, \gamma)$-partition $I_1 \cup I_2 \ldots \cup I_k$ for $(0, 1]$, Algorithm Approximate-Intervals(.) below gives the estimation for the number of items in each I_j if interval I_j has a number items to be large enough. Otherwise, those items in I_j can be ignored without affecting much of the approximation ratio. We have an adaptive way to do random samplings in a series of phases. Phase $t + 1$ doubles the number of random samples of phase t ($m_{t+1} = 2m_t$). For each phase, if an interval I_j shows sufficient number of items from the random samples, the number of items $C(I_j, S)$ in I_j can be sufficiently approximated by $\hat{C}(I_j, S)$. Thus, $\hat{C}(I_j, S)\pi_j$ also gives an approximation for the sum of the sizes of items in I_j. The sum $app_w = \sum_{I_j} \hat{C}(I_j, S)\pi_j$ for those intervals I_j with large number of samples gives an approximation for the total sum $\sum_{i=1}^{n} a_i$ of items in the input list. Let m_t denote the number of random samples in phase t. In the early stages, app_w is much smaller than $\frac{n}{m_t}$. Eventually, app_w will surpass $\frac{n}{m_t}$. This happens when m_t is more than $\frac{n}{\sum_{i=1}^{n} a_i}$ and app_w is close to the sum $\sum_{i=1}^{n} a_i$ of all items from the input list. This indicates that the number of random samples is sufficient for approximation algorithm. For those intervals with small number of samples, their items only need small fraction of bins to be packed. This process is terminated when ignoring all those intervals with none or small number of samples does not affect much of the accuracy of approximation. The algorithm gives up the process of random sampling when m_t surpasses n, and switches to use a deterministic way to access the input list, which happens when the total sum of the sizes of input items is $O(1)$. The lengthy analysis is caused by the multi-phases adaptive random samplings. We show two examples below.

Example 1: The input is a list of items such that there are three items of size 1, and the rest $n - 3$ items are of size 0.1 for a large integer n. Assume that ϵ is a positive constant to control the accuracy of approximation. After sampling a constant $\frac{100}{\epsilon}$ number of items, we observe all samples equal to 0.1 (with high probability). Thus, there are less than $\frac{\epsilon n}{20}$ items of size other than 0.1 with high probability by Chernoff bounds. We derive the approximate sum of total item sizes is $0.1n$, and output $\frac{0.1(1+\epsilon)n}{0.9}$ for the number bins for packing the input items, where the denominator 0.9 is based on the consideration that some bins for packing items of size 0.1 may waste up to 0.1 space. Although, there are small number of items of size 1, just ignoring those items of size 1 loses only a small accuracy of approximation. Therefore, the random sampling stops after sampling only $O(\frac{1}{\epsilon})$ items. We output an $(1 + \epsilon)$-approximation for the bin packing problem.

Example 2: The input is a list of items such that there are three items are of size 1, and the rest $n - 3$ items are of size $\frac{1}{n^2}$ for a large integer n. The number of random samples is doubled from one phase to next phase. After sampling $n^{0.9}$ items, in which there is no large items of size 1 with high probability, we still feel that those items of large size will greatly affect the total number bins. We have to continue use more random samples. Eventually, the number of random samples m_t is more than n. Thus, we switch to use a deterministic $O(n)$ time algorithm to compute the number of large items, the total sum of the sizes of items, and the total sum of the sizes of small items.

Algorithm Approximate-Intervals$(\varphi, \delta, \gamma, \theta, \alpha, P, n, S)$

Input: a parameter $\varphi \in (0,1)$, a small parameter $\theta \in (0,1)$, a failure probability upper bound α, a $(\varphi, \delta, \gamma)$ partition $P = I_1 \cup \ldots \cup I_k$ for $(0,1]$ with $\delta, \gamma \in (0,1)$, an integer n, a list S of n items $a_1, \ldots, a_n$ in $(0,1]$. Parameters $\varphi, \delta, \gamma, \theta$, and α do not depend on the number of items n.

Steps:

1. Phase 0:
2. Let $z := \xi_0 \log\log n$, where ξ_0 is a parameter such that $8(k+1)(\log n)g(\theta)^{z/2} < \alpha$ for all large n.
3. Let parameters $c_0 := \frac{1}{100}, c_2 := \frac{1}{3(1+\delta)c_0}, c_3 := \frac{\delta^4}{2(1+\delta)}, c_4 := \frac{8}{(1-\theta)(1-\delta)\varphi c_0}$, and $c_5 := \frac{12\xi_0}{(1-\theta)c_2c_3}$.
4. Let $m_0 := z$.
5. End of Phase 0.
6. Phase t:
7. Let $m_t := 2m_{t-1}$.
8. Sample m_t random items $a_{i_1}, \ldots, a_{i_{m_t}}$ from the input list S.
9. Let $d_j := |\{j : a_{i_j} \in I_j \ \ and\ 1 \le j \le m_t\}|$ for $j = 1, 2, \ldots, k$.
10. For each I_j,
11. if $d_j \ge z$,
12. then let $\hat{C}(I_j, S) := \frac{n}{m_t} d_j$ to approximate $C(I_j, S)$.
13. else let $\hat{C}(I_j, S) := 0$.
14. Let $app_w := \sum_{d_j \ge z} \hat{C}(I_j, S)\pi_j$ to approximate $\sum_{i=1}^{n} a_n$.
15. If $app_w \le \frac{c_5 n \log\log n}{c_0 m_t}$ and $m_t < n$ then enter Phase $t+1$.
16. else
17. If $m_t < n$
18. then let $app'_w := \sum_{d_j \ge z \ and\ j>1} \hat{C}(I_j, S)\pi_j$ to approximate $\sum_{a_i < \delta, 1 \le i \le n} a_i$.
19. else let $app_w := \sum_{i=1}^{n} a_i$ and $app'_w := \sum_{a_i < \varphi} a_i$.
20. Output app_w, app'_w and $\hat{C}(I_1, S)$ (the approximate number of items of size at least φ).
21. End of Phase t.

End of Algorithm

Lemma 9 uses several parameters $\varphi, \delta, \gamma, \alpha$ and θ that will be determined by the approximation ratio for the the bin packing problem. If the approximation ratio is fixed, they all become constants.

Lemma 9. *Assume that $\varphi, \delta, \gamma, \alpha$ and θ are parameters in $(0,1)$, and those parameters do not depend on the number of items n.. Then there exists a randomized algorithm described in Approximate-Intervals(.) such that given a list S of items of size $a_1, \ldots, a_n$ in the range $(0,1]$ and a $(\varphi, \delta, \gamma)$-partition for $(0,1]$, with probability at most α, at least one of the following statements is false after executing the algorithm:*

1. *For each* I_j *with* $\hat{C}(I_j, S) > 0$, $C(I_j, S)(1-\theta) \leq \hat{C}(I_j, S) \leq C(I_j, S)(1+\theta)$;

2. $\sum_{a_i \in I_j \text{ and } \hat{C}(I_j,S)=0} a_i \leq \frac{\delta^3}{2}(\sum_{i=1}^n a_i) + \frac{\gamma}{n}$;

3. $(1-\theta)(1-\delta)\varphi(\frac{\sum_{i=1}^n a_i}{2} - \frac{2\gamma}{n}) \leq app_w \leq (1+\theta)(\sum_{i=1}^n a_i)$;

4. *If* $\sum_{i=1}^n a_i \geq 4$, *then* $\frac{1}{4}(1-\theta)(1-\delta)\varphi(\sum_{i=1}^n a_i) \leq app_w \leq (1+\theta)(\sum_{i=1}^n a_i)$; *and*

5. *It runs in* $O(\frac{1}{(1-\theta)\delta^4 \log g(\theta)} \min(\frac{n}{\sum_{i=1}^n a_i}, n)(\log n) \log\log n)$ *time. In particular, the complexity of the algorithm is* $O(\min(\frac{n}{\sum_{i=1}^n a_i}, n)(\log n)\log\log n)$ *if* $\varphi, \delta, \gamma, \alpha$ *and* θ *are constants in* $(0,1)$.

Lemma 9 implies that with probability at least $1-\alpha$, all statements 1 to 5 are true. Due to the technical reason described at the end of section 2.1.2, we estimate the failure probability instead of the success probability.

Proof: Let $\xi_0, c_0, c_2, c_3, c_4$, and c_5 be parameters defined as those in the algorithm Approximate-Intervals(.). We use the uniform random sampling to approximate the number of items in each interval I_j in the $(\varphi, \delta, \gamma)$-partition.

Claim 9.1. Let Q_1 be the probability that the following statement is false:
(i) For each interval I_j with $d_j \geq z$, $(1-\theta)C(I_j, S) \leq \hat{C}(I_j, S) \leq (1+\theta)C(I_j, S)$.
Then for each phase in the algorithm, $Q_1 \leq (k+1) \cdot g(\theta)^{\frac{z}{2}}$.

Proof: Let $p_j = \frac{C(I_j,S)}{n}$. An element of S in I_j is sampled (by an uniform sampling) with probability p_j. Let $p' = \frac{z}{2m_t}$. For each interval I_j with $d_j \geq z$, we discuss two cases.

- Case 1. $p' \geq p_j$.

  In this case, $d_j \geq z \geq 2p'm_t \geq 2p_j m_t$. Note that d_j is the number of elements in interval I_j among m_t random samples $a_{i_1}, \ldots, a_{i_{m_t}}$ from S. By Theorem 5 (with $\theta = 1$), with probability at most $P_1 = g_2(1)^{p'm_t} \leq g_2(1)^{z/2} \leq g(1)^{z/2}$, there are at least $2p_j m_t$ samples are in from interval I_j.

- Case 2. $p' < p_j$.

  By Theorem 5, we have $\Pr[d_j > (1+\theta)p_j m_t] \leq g_2(\theta)^{p_j m_t} \leq g_2(\theta)^{p' m_t} \leq g_2(\theta)^{\frac{z}{2}} \leq g(\theta)^{\frac{z}{2}}$.

  By Theorem 4, we have $\Pr[d_j \leq (1-\theta)p_j m_t] \leq g_1(\theta)^{p_j m_t} \leq g_1(\theta)^{p' m_t} = g_1(\theta)^{\frac{z}{2}} \leq g(\theta)^{\frac{z}{2}}$.

  For each interval I_j with $d_j \geq z$ and $(1-\theta)p_j m_t \leq d_j \leq (1+\theta)p_j m_t$, we have $(1-\theta)C(I_j, S) \leq \hat{C}(I_j, S) \leq (1+\theta)C(I_j, S)$ by line 12 in Approximate-Intervals(.).

  There are $k = (\log n)$ intervals $I_1, \ldots, I_k$. Therefore, with probability at most $P_2 = k \cdot g(\theta)^{\frac{z}{2}}$, the following is false: For each interval I_j with $d_j \geq z$, $(1-\theta)C(I_j, S) \leq \hat{C}(I_j, S) \leq (1+\theta)C(I_j, S)$.

By the analysis of Case 1 and Case 2, we have $Q_1 \leq P_1 + P_2 \leq (k+1) \cdot g(\theta)^{\frac{z}{2}}$. Thus, the claim has been proven. ∎

Claim 9.2. Assume that $m_t \geq \frac{c_2 c_5 n \log\log n}{\sum_{i=1}^n a_i}$. Then right after executing Phase t in Approximate-Intervals(.), with probability at most $Q_2 = 2kg(\theta)^{\xi_0 \log\log n}$, the following statement is false:
(ii) For each interval I_j with $C(I_j, S) \geq c_3 \sum_{i=1}^n a_i$, A). $(1-\theta)C(I_j, S) \leq \hat{C}(I_j, S) \leq (1+\theta)C(I_j, S)$; and B). $d_j \geq z$.

Proof: Assume that $m_t \geq \frac{c_2 c_5 n \log\log n}{\sum_{i=1}^n a_i}$. Consider each interval I_j with $C(I_j, S) \geq c_3 \sum_{i=1}^n a_i$. We have that $p_j = \frac{C(I_j,S)}{n} \geq \frac{c_3 \sum_{i=1}^n a_i}{n}$. An element of S in I_j is sampled with probability p_j. By Theorem 5 and Theorem 4, we have

$$\Pr[d_j < (1-\theta)p_j m_t] \leq g_1(\theta)^{p_j m_t} \leq g_1(\theta)^{c_2 c_3 c_5 \log\log n} \leq g(\theta)^{\xi_0 \log\log n}. \quad (1)$$

$$\Pr[d_j > (1+\theta)p_j m_t] \leq g_2(\theta)^{p_j m_t} \leq g_2(\theta)^{c_2 c_3 c_5 \log\log n} \leq g(\theta)^{\xi_0 \log\log n}. \quad (2)$$

Therefore, with probability at most $2kg(\theta)^{\xi_0 \log\log n}$, the following statement is false:
For each interval I_j with $C(I_j,S) \geq c_3 \sum_{i=1}^n a_i$, $(1-\theta)C(I_j,S) \leq \hat{C}(I_j,S) \leq (1+\theta)C(I_j,S)$.
If $d_j \geq (1-\theta)p_j m_t$, then we have

$$\begin{aligned} d_j &\geq (1-\theta)\frac{C(I_j,S)}{n} m_t \\ &\geq (1-\theta)\frac{(c_3 \sum_{i=1}^n a_i)}{n} \cdot \frac{c_2 c_5 n \log\log n}{\sum_{i=1}^n a_i} \\ &= (1-\theta) c_2 c_3 c_5 \log\log n \\ &\geq \xi_0 \log\log n = z. \end{aligned}$$

∎

Claim 9.3. The total sum of the sizes of items in those I_js with $C(I_j,S) < c_3 \sum_{i=1}^n a_i$ is at most $\frac{\delta^3}{2}(\sum_{i=1}^n a_i) + \frac{\gamma}{n}$.
Proof: By definition 7, we have $a_j = \varphi(1-\delta)^{j-1}$ for $j = 1, \ldots, k-1$. We have that

- the sum of sizes of items in I_k is at most $n\frac{\gamma}{n^2} = \frac{\gamma}{n}$,
- for each interval I_j with $C(I_j,S) < c_3 \sum_{i=1}^n a_i$, the sum of sizes of items in I_j is at most $(c_3 \sum_{i=1}^n a_i) a_{j-1} \leq (c_3 \sum_{i=1}^n a_i)\varphi(1-\delta)^{j-2}$ for $j \in (1,k)$, and
- the sum of sizes in I_1 is at most $c_3 \sum_{i=1}^n a_i$.

The total sum of the sizes of items in those I_js with $C(I_j,S) < c_3 \sum_{i=1}^n a_i$ is at most $(c_3 \sum_{i=1}^n a_i) + \sum_{j=2}^k (c_3 \sum_{i=1}^n a_i)\varphi(1-\delta)^{j-2}) + n \cdot \frac{r}{n^2} \leq (c_3 \sum_{i=1}^n a_i) + \frac{c_3\varphi}{\delta}(\sum_{i=1}^n a_i) + \frac{\gamma}{n} \leq \frac{\delta^3}{2}(\sum_{i=1}^n a_i) + \frac{\gamma}{n}$. ∎

Claim 9.4. Assume that at the end of phase t, for each I_j with $\hat{C}(I_j,S) > 0$, $C(I_j,S)(1-\theta) \leq \hat{C}(I_j,S) \leq C(I_j,S)(1+\theta)$; and $d_j \geq z$ if $C(I_j,S) \geq c_3 \sum_{i=1}^n a_i$. Then $(1-\theta)(1-\delta)\varphi(\frac{\sum_{i=1}^n a_i}{2} - \frac{2\gamma}{n}) \leq app_w \leq (1+\theta)(\sum_{i=1}^n a_i)$ at the end of phase t.
Proof: By the assumption of the claim, we have $app_w = \sum_{d_j \geq z} \hat{C}(I_j,S)\pi_j \leq (1+\theta)\sum_{i=1}^n a_i$. For each interval I_j with $j \neq k$ and $j > 1$, we have $C(I_j,S)\pi_j \geq (1-\delta)\sum_{a_i \in I_j} a_i$ by the definition of (φ,δ,γ)-partition. It is easy to see that $C(I_1,S)\pi_1 \geq \varphi \sum_{a_i \in I_1} a_i$ by the definition of (φ,δ,γ)-partition. Thus,

$$C(I_j,S)\pi_j \geq (1-\delta)\varphi \sum_{a_i \in I_j} a_i \quad \text{for} \quad j \neq k. \tag{3}$$

We have the following inequalities:

$$\begin{aligned} app_w &= \sum_{d_j \geq z} \hat{C}(I_j,S)\pi_j \quad \text{(by line 15 in Approximate-Intervals(.))} \\ &\geq (1-\theta)\sum_{d_j \geq z} C(I_j,S)\pi_j \\ &\geq (1-\theta)\sum_{d_j \geq z, j \neq k} C(I_j,S)\pi_j \\ &\geq (1-\theta)(1-\delta)\varphi \sum_{d_j \geq z, j \neq k} \left(\sum_{a_i \in I_j} a_i \right) \quad \text{(by inequality (3))} \\ &\geq (1-\theta)(1-\delta)\varphi(\sum_{i=1}^n a_i - \sum_{d_j < z} \sum_{a_i \in I_j} a_i - \sum_{a_i \in I_k} a_i) \end{aligned}$$

$$\geq \quad (1-\theta)(1-\delta)\varphi(\sum_{i=1}^{n} a_i - (\frac{\delta^3}{2}(\sum_{i=1}^{n} a_i) + \frac{\gamma}{n}) - n \cdot \frac{\gamma}{n^2}) \quad \text{(by Claim 9.3)}$$

$$\geq \quad (1-\theta)(1-\delta)\varphi(\frac{\sum_{i=1}^{n} a_i}{2} - \frac{2\gamma}{n}).$$

∎

Claim 9.5. With probability at most $Q_5 = (k+1) \cdot (\log n) g(\theta)^{\frac{z}{2}}$, the following facts are not all true:

A. For each phase t with $m_t < \frac{2c_2c_5 n \log\log n}{\sum_{i=1}^{n} a_i}$, the condition $app_w \leq \frac{c_5 n \log\log n}{c_0 m_t}$ in line 15 of the algorithm is true.

B. If $\sum_{i=1}^{n} a_i \geq 4$, then the algorithm stops before $m_t > \frac{2c_4c_5 n \log\log n}{\sum_{i=1}^{n} a_i}$.

C. If $\sum_{i=1}^{n} a_i \leq 4$, then it stops before or at phase t in which the condition $m_t \geq n$ first becomes true.

Proof: By Claim 9.1, with probability at most $(k+1) \cdot g(\theta)^{\frac{z}{2}}$, the statement i of Claim 9.1 is false for a fixed m. The number of phases is at most $\log n$ since m_t is double at each phase. With probability $(k+1) \cdot (\log n) \cdot g(\theta)^{\frac{z}{2}}$, the statement i of Claim 9.1 is false for each phase t with $m_t \leq n$. Assume that statement i of Claim 9.1 is true for all phases t with $m_t \leq n$.

Statement A. Assume that $m_t < \frac{2c_2c_5 n \log\log n}{\sum_{i=1}^{n} a_i}$. We have $\frac{n}{m_t} > \frac{n}{\frac{2c_2c_5 n \log\log n}{\sum_{i=1}^{n} a_i}} = \frac{\sum_{i=1}^{n} a_i}{2c_2c_5 \log\log n}$. Therefore, $\sum_{i=1}^{n} a_i < (\frac{n}{m_t}) 2c_2c_5 \log\log n = \frac{2c_2c_5 n \log\log n}{m_t}$. By Claim 9.4, $app_w \leq (1+\theta)\sum_{i=1}^{n} a_i$. Since $(1+\theta) < \frac{1}{2c_2c_0}$ (by line 3 in Approximate-Intervals(.)), we have

$$app_w \leq (1+\theta)\sum_{i=1}^{n} a_i \leq \frac{1}{2c_2c_0}\sum_{i=1}^{n} a_i < \frac{1}{2c_2c_0} \cdot \frac{2c_2c_5 n \log\log n}{m_t} = \frac{c_5 n \log\log n}{c_0 m_t}.$$

Statement B. The variable m_t is doubled in each new phase. Assume that the algorithm enters phase t with $\frac{c_4c_5 n \log\log n}{\sum_{i=1}^{n} a_i} \leq m_t \leq \frac{2c_4c_5 n \log\log n}{\sum_{i=1}^{n} a_i}$. We have $\frac{n}{m_t} \leq \frac{n}{\frac{c_4c_5 n \log\log n}{\sum_{i=1}^{n} a_i}} = \frac{\sum_{i=1}^{n} a_i}{c_4c_5 \log\log n}$. Since $\sum_{i=1}^{n} a_i \geq 4$, $(\frac{\sum_{i=1}^{n} a_i}{2} - \frac{\gamma}{n}) \geq \frac{\sum_{i=1}^{n} a_i}{4}$. By Claim 9.4, app_w is at least $\frac{(1-\theta)(1-\delta)\varphi}{4}\sum_{i=1}^{n} a_i$. Since $\frac{(1-\theta)(1-\delta)\varphi}{4} > \frac{1}{c_0c_4}$, we have $app_w > \frac{c_5 n \log\log n}{c_0 m}$, which makes the condition at line 15 in Approximate-Intervals(.) be false. Thus, the algorithm stops at some stage t with $m_t \leq \frac{2c_4c_5 n \log\log n}{\sum_{i=1}^{n} a_i}$ by the setting at line 15 in Approximate-Intervals(.).

Statement C. It follows from statement A and the setting in line 15 of the algorithm. ∎

Claim 9.6. The complexity of the algorithm is $O(\frac{1}{(1-\theta)\delta^4 \log g(\theta)} \min(\frac{n}{\sum_{i=1}^{n} a_i}, n)(\log n)\log\log n)$. In particular, the complexity is $O(\min(\frac{n}{\sum_{i=1}^{n} a_i}, n)(\log n)\log\log n)$ if $\varphi, \delta, \gamma, \alpha$ and θ are constants in $(0,1)$.

Proof: By the setting in line 3 in Approximate-Intervals(.), we have

$$\begin{aligned} c_2c_5 &= \frac{1}{3(1+\delta)c_0} \cdot \frac{12\xi_0}{(1-\theta)c_2c_3} \\ &= \frac{4\xi_0}{(1+\delta)\cdot c_0 \cdot (1-\theta)\cdot \frac{1}{3(1+\delta)c_0} \cdot \frac{\delta^4}{2(1+\delta)}} \\ &= \frac{24\xi_0(1+\delta)}{(1-\theta)\delta^4}. \end{aligned}$$

In order to satisfy the condition $8(k+1)(\log n)g(\theta)^{z/2} < \alpha$ for all large n at line 2 in Approximate-Intervals(.), we can let $\xi_0 = \frac{8}{\log g(\theta)}$.

Since m_t is doubled every phase, the total number of phases is at most $\log n$. The computational time complexity in statement 5 of the algorithm follows from Claim 9.5. ∎

As m_t is doubled each new phase in Approximate-Intervals(.), the number of phases is at most $\log n$. With probability at most $(\log n)(Q_1 + Q_2) + Q_5 \leq \alpha$ (by line 2 in Approximate-Intervals(.)), at least one of the statements (i) in Claim 9.1, (ii) in Claim 9.2, A, B, C in Claim 9.5 is false.

Assume that the statements (i) in Claim 9.1, (ii) in Claim 9.2, A, B, and C in Claim 9.5 are all true.

For an interval I_j, $\hat{C}(I_j, S) > 0$ if and only if $d_j \geq z$ by lines 10 to 13 in Approximate-Intervals(.). Therefore, statement 1 of the lemma follows from Claim 9.1.

If Approximate-Intervals(.) stops at $m_t < n$, then $m_t \geq \frac{2c_2c_5 n \log\log n}{\sum_{i=1}^{n} a_i}$ by statement A in Claim 9.5. For each interval I_j with $C(I_j, S) \geq c_3 \sum_{i=1}^{n} a_i$, we have $d_j \geq z$, which implies $\hat{C}(I_j, S) > 0$. Statement 2 of Lemma 9 follows from Claim 9.3 and statement (ii) of Claim 9.2.

Statement 3 follows from Claim 9.4. The condition of Statement 4 implies $n \geq 4$. Statement 4 follows from Statement 3. Statement 5 for the running time follows from Claim 9.6.

Thus, with probability at most α, at least one of the statements 1 to 5 is false. ∎

4. Main Results

We list the main results that we achieve in this paper. The proof of Theorem 10 is shown in Section 6.3.

Theorem 10 (Main). *Approximate-Bin-Packing(.) is a randomized approximation scheme for the bin packing problem such that given an arbitrary $\tau \in (0, 1)$ and a list of items $S = a_1, \ldots, a_n$ in $(0, 1]$ for the bin packing problem, it gives an approximation app with $Opt(S) \leq app \leq (1 + \tau)Opt(S) + 1$ in $O(\frac{n(\log n)(\log\log n)}{\sum_{i=1} a_i} + (\frac{1}{\tau})^{O(\frac{1}{\tau})})$ time with probability at least $\frac{3}{4}$.*

We show a lower bound for those bin packing problems with bounded sum of sizes $\sum_{i=1}^{n} a_i$. The lower bound always matches the upper bound.

Theorem 11. *Assume $f(n)$ is a nondecreasing unbounded function from N to N with $f(n) = o(n)$. Every randomized $(2-\epsilon)$ approximation algorithm for bin packing problems in $\sum(f(n))$ needs $\Omega(\frac{n}{f(n)})$ time, where ϵ is an arbitrary small constant in $(0, 1)$.*

Proof: Since $f(n)$ is unbounded, assume n is large enough such that

$$(f(n) + 2)(2 - \epsilon) < 2(f(n) - 2). \tag{4}$$

We design two input list of items.

The first list contains $m = 2(f(n) - 2))$ elements of size $\frac{1}{2} + \delta$, where $\delta = \frac{1}{2(f(n)-2)}$. The rest $n - m$ items are of the same size $\gamma = \frac{1}{n-m} = o(1)$. We have $m(\frac{1}{2} + \delta) + (n - m)\gamma = 2(f(n) - 2)(\frac{1}{2} + \frac{1}{2(f(n)-2)}) + 1 = f(n)$. Therefore, the first list is a bin packing problem is in $\sum(f(n))$.

The second list contains $n - f(n)$ elements of size γ and the rest $f(n)$ items are of size equal to $1 - \tau$, where $\tau = \frac{(n-f(n))\gamma}{f(n)} = o(1)$. We have $f(n)(1 - \tau) + (n - f(n))\gamma = f(n)$. The second list is also a bin packing problem is in $\sum(f(n))$.

Both γ and τ are small. Packing the first list needs at least $2(f(n) - 2)$ bins. Packing the second list only needs at most $f(n) + 2$ bins since two bins of size one is enough to pack those items of size τ.

Assume that an algorithm only has computational time $o(\frac{n}{f(n)})$ for computing $(2 - \epsilon)$-approximation for bin packing problems in $\sum(f(n))$. The algorithm has an $o(1)$ probability to

access at least one item of size at least $\frac{1}{2}$ in both lists. Therefore, the two lists have the same output for approximation by the same randomized algorithm. For the second list, the output for the number of bins should be at most $(f(n)+2)(2-\epsilon)$. By inequality (4), it is impossible to pack the first list items. This brings a contradiction. ∎

Corollary 12. *There is no* $o(\frac{n}{\sum_{i=1}^{n} a_i})$ *time randomized approximation scheme algorithm for the bin packing problem.*

Proof: It follows from Theorem 11. ∎

5. Generalization of the Deterministic Algorithm

In this section, we generalize the existing deterministic algorithm [11] to handle the bin packing problem with multiple sizes of bins. The bin packing problem is under a more general version that allows different size of bins with different weights (costs). The results of this section are used as submodules in both sublinear time algorithms and streaming algorithms.

Definition 13.

- For an item y and an integer h, define y^h to be h copies of item y.
- A type T_i of a bin of size s is represented by $(a_1^{b_{1,i}}, \ldots, a_t^{b_{t,i}})$, which satisfies $\sum_{j=1}^{t} b_{j,i} a_i \leq s$. A bin of type T_i can pack $b_{1,i}$ items of size $a_1, \ldots,$, and $b_{t,i}$ items of size a_t. We use w_{T_i} to represent the weight of a bin of type T_i.

It is easy to see that an optimal bin packing with (c, η, k)-related bins only uses bins with $s_{i_1} < s_{i_2} < \ldots < s_{i_k}$ with $w_{i_1} < w_{i_2} < \ldots < w_{i_k}$. The classical bin packing problem only has one kind of bins of size 1. It is the bin packing problem with the $(1,1,1)$-related bins. In the rest of this paper, a bin packing problem without indicating (c, η, k)-related bins means the classical bin packing problem.

Lemma 14. *Assume that* c*,* η*, and* k *are constants. Assume that* δ *is a constant. Given a bin packing problem with* (c, η, k)*-related bins for* $B = \{a_1^{n_1}, \ldots, a_m^{n_m}\}$ *with each* $a_i \geq \delta$*, there is a* $m^{O(\frac{1}{\delta})}$ *time algorithm to give a solution* $(x_1, \ldots, x_q)$ *with at most* $Opt_{c,\eta,k}(B) + \sum_{i=1}^{q} w_{T_i}$*, where* x_i *is the number of bins of type* T_i*, and* q *is the number of types to pack items of sizes in* $\{a_1, \ldots, a_m\}$ *with* $q \leq km^{\frac{1}{\delta}}$*.*

Proof: Since a_i is at least δ, the number of items in each bin is at most $\frac{1}{\delta}$. Therefore, the number of types of bins is at most $km^{\frac{1}{\delta}}$. Let $T_1, \ldots, T_q$ be the all of the possible types of bins to pack the items of size $a_1, \ldots, a_m$.

Let x_i be the number of bins with type T_i. We define the linear programming conditions:

$$\min \sum_{i=1}^{q} w_{T_i} x_i \qquad \text{subject to} \tag{5}$$

$$\sum_{i=1}^{q} b_{j,i} x_i \geq n_j \quad \text{for } j = 1, 2, \ldots, m \tag{6}$$

$$x_i \geq 0. \tag{7}$$

After obtaining the optimal solution $(x_1^*, \ldots, x_q^*)$ of the linear programming, the algorithm outputs $(x_1, \ldots, x_q) = (\lceil x_1^* \rceil, \ldots, \lceil x_q^* \rceil)$. Since $\lceil x_i^* \rceil \leq x_i^* + 1$, the cost for $(x_1, \ldots, x_q)$ is at most $Opt_{c,\eta,k}(B) + \sum_{i=1}^{q} w_{T_i}$.

Algorithm Pack-Large-Items(c, η, k, B)
Input: parameters c, η, k and a list $B = \{a_1^{n_1} \ldots, a_m^{n_m}\}$ to be packed in (c, η, k) related bins.
Output: an approximation for $Opt_{c,\eta,k}(B)$.
Steps:
Solve the linear programming (5)-(7) for $x_1^*, \ldots, x_q^*$.
Let $x_i = \lceil x_i^* \rceil$ for $i = 1, \ldots, q$.
Output $(x_1, \ldots, x_q)$.
End of Algorithm

∎

With a constant ϵ to control the approximation ratio, we define the following constants for Lemma 20. We will also define a threshold δ to control the size of large items. Let

$$\mu := \frac{\epsilon\delta\eta}{15}, \tag{8}$$

$$\epsilon_1 := \frac{\epsilon}{\epsilon + 2}, \text{ and} \tag{9}$$

$$m := \frac{18}{\delta\eta}\left\lceil \epsilon_1^{-2} \right\rceil. \tag{10}$$

Lemma 15. *Assume that c, η, and k are positive constants, and ϵ and δ are constants in $(0, 1)$. Assume that the input list is S for bin packing problem with (c, η, k)-related bins and the size of each item in S is at least δ. Let ϵ be a constant in $(0, 1)$. The constants δ, μ, ϵ_1, and m are given according to equations (8) to (10). Let $h = \lfloor \frac{n}{m} \rfloor$. Then there exists an $O(n)$ time algorithm that gives an approximation app with $Opt_{c,\eta,k}(S) \leq app \leq (1 + \epsilon)Opt_{c,\eta,k}(S)$ for all large n, where $n = |S|$.*

Proof: Assume that $a_1 \leq a_2 \leq \ldots \leq a_n$ is the increasing order of all input elements at least δ with $n' = |S_{\geq\delta}|$. Let $L_0 = a_1 \leq a_2 \leq \ldots \leq a_n$. We partition $a_1 \leq a_2 \leq \ldots \leq a_n$ into $A_1 y_1 A_2 y_2 \ldots A_m y_m R$ such that each A_i has exactly $h - 1$ elements and R has less than h elements.

Using algorithm the classical algorithm, we can find the ih-th element y_i each in $O(n)$ time.

Consider the bin packing problems: $L_1 = y_1^h y_2^h \ldots y_m^h$. We show that there is a small difference between the results of two bin packing problems for L_0 and L_1.

1) Assume that L_0 has a bin packing solution. It can be converted into a solution for L_1 via an adaption to that of L_0 (see Definition 13) with a small number of additional bins.

Use the lots for the elements between y_i and y_{i+1} in L_0 to store the elements of y_is, there are at most $2h$ y_is left. Therefore, we only have at most $2h$ elements left. The number of bins for packing those left items is at most $2h$, which cost at most $2h$ since 1 is the maximal cost of one bin.

2) Assume that L_1 has a bin packing solution. It can be converted into a solution for L_0 with a small number of additional bins.

We use the lots for y_i to store the elements between y_{i-1} and y_i. We have at most $2h$ elements left, which cost at most $2h$ since 1 is the maximal cost of one bin.

The optimal number bins $Opt_{c,\eta,k}(L_0)$ for packing L_0 is at least $mh\delta$, which have cost at least $mh\delta\eta$. Therefore, we have

$$Opt_{c,\eta,k}(L_0) \geq mh\delta\eta \tag{11}$$

Let $App(L_0)$ be an approximation for L_0 and $App(L_1)$ be an approximation for L_1. We can obtain an $(1 + \epsilon/2)$-approximation $App(L_1)$ for packing L_1 by Lemma 14. We have that

$$\begin{aligned} App(L_0) &= App(L_1) + 2h \\ &\leq (1 + \epsilon/2)Opt_{c,\eta,k}(L_1) + 2h \\ &\leq ((1 + \epsilon/2)(Opt_{c,\eta,k}(L_0) + 2h) + 2h \end{aligned}$$

$$
\begin{aligned}
&\leq ((1+\epsilon/2)Opt_{c,\eta,k}(L_0) + 6h \\
&\leq (1+\epsilon/2)Opt_{c,\eta,k}(L_0) + Opt_{c,\eta,k}(L_0)(\frac{6h}{mh\delta\eta}) \quad \text{(by inequality (11))} \\
&= (1+\epsilon/2)Opt_{c,\eta,k}(L_0) + Opt_{c,\eta,k}(L_0)\frac{6}{m\delta\eta} \\
&\leq (1+\epsilon/2)Opt_{c,\eta,k}(L_0) + Opt_{c,\eta,k}(L_0)(\epsilon/2) \qquad \text{(by equations (8) to (10).)} \\
&\leq (1+\epsilon)Opt_{c,\eta,k}(L_0).
\end{aligned}
$$

By the analysis at case 2), if $App(L_1) \geq Opt_{c,\eta,k}(L_1)$, we also have that the cost $App(L_1) + 2h$ is enough to pack all items in L_0. Therefore,

$$App(L_0) \geq Opt_{c,\eta,k}(L_0). \tag{12}$$

For a bin b_i, let $l(b_i)$ be the sum of sizes of items packed in it.

Algorithm Packing(L_0)
Input: a list $L_0 := \{a_1 \dots a_m\}$
Output: an approximation for $Opt_{c,\eta,k}(L_0)$.
Steps:
Find the ih-th element y_i in L_0 for $i = 1, \dots, m$.
Let $L_1 := y_1^h y_2^h \dots y_m^h$.
Let $(x_1, \dots, x_q)$:=Pack-Large-Items$(1, 1, 1, L_1)$ (see Lemma 14).
Let $App(L_1) := \sum_{i=1}^{q} w_{T_i} x_i$.
Convert $App(L_1)$ to $App(L_0)$ according to equation (12).
Let $B = b_1, \dots, b_u$ be the list of bins used for packing (each b_i has $l(b_i)$ available).
Output $App(L_0)$, and list B of bins.
End of Algorithm
We note that the list of bins $b_1, \dots, b_u$ with their used space $l(b_i)$ for each bin can be computed in $O(n)$ time from the conversion based on $(x_1, \dots, x_q)$ for q types $T_1, \dots, T_q$.

∎

Lemma 16 ([11]). *Let β be a constant in $(0, 1)$. Then there exists an $O(n)$ time algorithm that gives an approximation app for packing S with $Opt(S) \leq app \leq (1+\beta)Opt(S) + 1$ for all large n.*

Proof: The bin packing problem is the same as the regular bin packing problem that all bins are of the same size 1. The problem is to minimize the total number bins to pack all items. We consider the approximation to pack the small items after packing large items.

Assume that the input list is S for bin packing problem. Let $S_{<\delta}$ be the items of size less than δ, and $S_{\geq\delta}$ be the items of size at least δ. Let δ be a constant with $\delta \leq \frac{\beta}{4}$.

Algorithm Linear-Time-Packing(n, S)
Input: A list of items $S = a_1 \dots a_n$ and its number of items n.
Output: an approximation for $Opt(S)$.
Steps:

1. Let $App(S_{\geq\delta})$ and the bin list $b_1, \dots, b_u$ be the output from calling Packing$(S_{\geq\delta})$ (see Lemma 15).
2. for $i = 1$ to u
3. If $l(b_i) \leq 1 - \delta$
4. Fill items from $S_{<\delta}$ into b_i until less than δ space left in b_i or all items in $S_{<\delta}$ are packed.
5. If there are some items of size less than δ left

6. Then pack them into some bins so that at most one bin having more than δ space used.

7. Output the total number of bins used.

End of Algorithm

Assume that an optimal solution of a bin packing problem has two types of bins. Each of the first type contains at least one item of size at least δ, and each of the second type only contain items of size less than δ. Let V_1 be the set of first type bins, and V_2 be the set of all second type bins. Let $U = App(S_{\geq\delta})$ be an $(1+\beta)$-approximation for packing the first type of items. We have $|U| \leq (1+\beta)|V_1|$.

Fill all items into those bins in U so that each bin has less than δ left. Put all of the items less than δ into some extra bins, and at most one of them has more than δ space left.

Case 1. If U can contain all items, we have that $|U| \leq (1+\beta)|V_1| \leq (1+\beta)|V_1 \cup V_2| = (1+\beta)Opt(S)$.

Case 2. There is a bin beyond those in U is used. Let U' be all bins without more than δ space left. We have that $|U'| \leq \frac{|V_1 \cup V_2|}{(1-\delta)} \leq (1+\beta)|V_1 \cup V_2| = (1+\beta)Opt(S)$. Therefore, the approximate solution is at most $(1+\beta)|V_1 \cup V_2| + 1 = (1+\beta)Opt(S) + 1$.

∎

6. Randomized Offline Algorithm

In this section, we present sublinear time approximation schemes in the offline model.

6.1. Selecting Items from A List

In this section, we show how a randomized algorithm to select some crucial items from a list. Those elements are used for converting the packing large items into linear programming as described in Section 5.

In order to let linear programming have a small number of cases, the ih-th elements are selected for $i = 1, 2, \ldots, m$, where the large items are grouped into m groups with h items each. The approximate ih-th elements (for $i = 1, \ldots, m$) have similar performance as the exact ih-th elements in the linear programming method. The approximate ih-th elements (for $i = 1, \ldots, m$) can be obtained via sampling small number of items. The ih-th element among the large items is approximated by the ih-th element among the random samples from large items in the input list. The detail of the algorithm is given at Select-Crucial-Items(.).

For a finite set A, let $|A|$ be the number of elements in A. For a list L of items $a_1, \ldots, a_n$, let $|L| = n$.

Definition 17. Assume that $L = a_1, \ldots, a_n$ is the list of real numbers, and x is an integer.

- Define $\text{Rank}(x, L)$ in $a_1, \ldots, a_n$ to be the interval $[a, b]$ such that $|\{i : a_i < x\}| = a - 1$ and $|\{i : a_i \leq x\}| = b$. Define $\text{minRank}(x, L)$ to be a and $\text{maxRank}(x, L)$ to be b.
- Define $\text{Rank}_\delta(x, L)$ in $a_1, \ldots, a_n$ to be the interval $[a, b]$ such that $|\{i : a_i < x \text{ and } a_i \geq \delta\}| = a - 1$ and $|\{i : a_i \leq x \text{ and } a_i \geq \delta\}| = b$. Define $\text{minRank}_\delta(x, L)$ to be a and $\text{maxRank}_\delta(x, L)$ to be b.
- $L[s, t] = a_s, a_{s+1}, \ldots, a_t$ for $0 < s \leq t \leq n$.

Definition 18. Assume that S is a list of items for a bin packing problem and δ is a real number. Define $S_{<\delta}$ to be the sublist of the items of size less than δ in S, and $S_{\geq\delta}$ to be the sublist of the items of size at least δ in S.

By the definitions 17 and 18, we have

$$\begin{aligned} \text{minRank}_\delta(x, L) &= \text{minRank}(x, L_{\geq\delta}), & (13)\\ \text{maxRank}_\delta(x, L) &= \text{maxRank}(x, L_{\geq\delta}), \quad \text{and} & (14)\\ \text{Rank}_\delta(x, L) &= \text{Rank}(x, L_{\geq\delta}). & (15) \end{aligned}$$

Let m be a parameter at most n and let

$$h = \left\lfloor \frac{n}{m} \right\rfloor. \tag{16}$$

Let the sorted input list is partitioned into $K_1K_2 \ldots K_mR$ such that $|K_1| = |K_2| = \ldots = |K_m| = h$, and $0 \leq |R| < h$.

Algorithm Select-Crucial-Items(m, α, μ, X)

Input: two constants α and μ in $(0, 1)$, an integer parameter m at least 2, and a list $X = x_1, x_2, \ldots,$ is a finite list of random elements in A.

Steps:

1. Select $\gamma = \frac{\mu}{4m}$.
2. Select constant c_0 and $u = \left\lceil \frac{c_0 \log m}{\gamma^2} \right\rceil$ such that $2me^{-\frac{\gamma^2 u}{3}} < \alpha$ and $3 \leq \gamma u$.
3. If $v < u$ or $|X| < u$, then output $\emptyset$ and stop the algorithm.
4. Let $p_i := \frac{i}{m}$ for $i = 1, \ldots, m$.
5. Let y_i $(i = 1, \ldots, m)$ be the least element x_j such that $|\{t : x_t \text{ is in } X[1, u] \text{ and } x_t \leq x_j\}| \geq \lceil p_i u \rceil$.
6. Output $(y_1, \ldots, y_m)$.

End of Algorithm

Lemma 19 shows the performance of the algorithm Algorithm Select-Crucial-Items(.). It is a step to convert the step for packing large items into a dynamic programming method. When the input list of items is S, the list A in Lemma 19 is the sublist $S_{\geq\delta}$ of all items of S with size at least δ, which will be specified in the full algorithm. The random items X is generated from the subset of all random items of sizes at least δ in a set of random items in S.

Lemma 19. *Let μ and α be positive constants in $(0, 1)$. Assume that A is an input list of n numbers of size at least δ with $n \geq \frac{3(m+1)^2}{\mu}$. Then the algorithm Select-Crucial-Items(.) runs in $O(\frac{m^2(\log m)^2)}{\mu^2})$ time such that given a list X of at least $\frac{c_1 m^2 \log m}{2}$ random elements from A, it generates elements $y_1 \leq \ldots \leq y_m$ from the input list such that $\Pr[\text{Rank}(y_i, A) \cap [ih - \mu h, ih + \mu h]] = \emptyset$ for at least one $i \in \{1, \ldots, m\}] \leq \alpha$, where $c_1 = 16c_0$, and c_0 is the constant defined in Select-Crucial-Items(.), and m is an integer at most n.*

Proof: The algorithm probabilistic performance is analyzed with Chernoff bounds. Note that the number of items n in A is not an input of this algorithm. We only use it in the analysis, but not in the algorithm. Without loss of generality, we assume $|X| = u$, where u is defined in statement 2 in the Algorithm Select-Crucial-Items(.).

According to the algorithm $u = \left\lceil \frac{c_0}{\gamma^2} \log m \right\rceil = \left\lceil \frac{16c_0 m^2 \log m}{\mu^2} \right\rceil = \left\lceil \frac{c_1 m^2 \log m}{\mu^2} \right\rceil$. We assume the number of random items in X is at least u. By the equation (16) and the fact $m \leq n$, we have

$$\begin{aligned} h &\leq \frac{n}{m} \leq h + 1 \leq 2h, \quad \text{and} & (17)\\ \frac{h}{n} &\leq \frac{1}{m}. & (18) \end{aligned}$$

By statement 1 in Select-Crucial-Items(.) and inequality (17), we have $\frac{n}{m} \leq 2h$ and

$$2\gamma \leq \frac{\mu}{2m} \leq \frac{\mu h}{n}. \tag{19}$$

Assume $\text{maxRank}(y_i, A) < ih - \mu h$. We have that

$$\begin{aligned}
\frac{\text{maxRank}(y_i, A)}{n} &< \frac{ih - \mu h}{n} && (20)\\
&= \frac{ih}{n} - \frac{\mu h}{n} && (21)\\
&\leq \frac{i}{m} - \frac{\mu h}{n} \quad \text{(by inequality (18))} && (22)\\
&\leq p_i - \frac{\mu h}{n}. && (23)
\end{aligned}$$

Let $p_i' := p_i - \frac{\mu h}{n} > \frac{\text{maxRank}(y_i, A)}{n}$ (by inequality (23)). By Corollary 6, with probability at most $e^{-\frac{\gamma^2 u}{3}}$, we have $|\{j : x_j \in X[1, u] \text{ and } x_j \leq y_i\}|$ to be at least

$$\begin{aligned}
(\frac{\text{maxRank}(y_i, A)}{n} + \gamma)u &< p_i' u + \gamma u\\
&= (p_i - (\frac{\mu h}{n}))u + \gamma u\\
&= p_i u - (\frac{\mu h}{n} - \gamma)u\\
&\leq p_i u - \gamma u \text{ (by inequality (19))}\\
&\leq \lceil p_i u \rceil
\end{aligned}$$

Assume $\text{minRank}(y_i, A) > ih + \mu h$. We have that

$$\begin{aligned}
\frac{\text{minRank}(y_i, A)}{n} &> \frac{ih + \mu h}{n} && (24)\\
&= \frac{ih}{n} + \frac{\mu h}{n} && (25)\\
&\geq \frac{i}{m} - \frac{i}{n} + \frac{\mu h}{n} \quad \text{(by equation (16))} && (26)\\
&\geq p_i - \frac{i}{n} + \frac{\mu h}{n}. && (27)
\end{aligned}$$

Note that the transition from inequality (25) to inequality (26) is due to equation (16)), which implies $h \geq \frac{n}{m} - 1$ and $\frac{h}{n} \geq \frac{1}{m} - \frac{1}{n}$.

Let $p_i'' := p_i - \frac{i}{n} + \frac{\mu h}{n} < \frac{\text{minRank}(y_i, A)}{n}$ (by inequality (27). Note that p_i is defined at line 4 in Algorithm Select-Crucial-Items(.). By Lemma 6, with probability at most $P_{1,i} = e^{-\frac{\gamma^2 u}{3}}$, we have $|\{j : x_j \in X[1, u] \text{ and } x_j \leq y_i\}|$ to be at most

$$\begin{aligned}
(\frac{\text{maxRank}(y_i, A)}{n} - \gamma)u &\geq p_i'' u - \gamma u && (28)\\
&= (p_i - \frac{i}{n} + \frac{\mu h}{n})u - \gamma u && (29)\\
&\geq p_i u + (\frac{\mu h}{n} - \frac{i}{n} - \gamma)u && (30)\\
&\geq p_i u + (\frac{\mu h}{n} - \frac{m}{n} - \gamma)u && (31)
\end{aligned}$$

$$\geq \quad p_i u + (\frac{\mu h}{3n} - \frac{m}{n})u + (\frac{2\mu h}{3n} - \gamma)u \tag{32}$$

$$\geq \quad p_i u + 0 + (\frac{2\mu h}{3n} - \gamma)u \tag{33}$$

$$\geq \quad p_i u + (\frac{4\gamma}{3} - \gamma)u \tag{34}$$

$$\geq \quad p_i u + \frac{\gamma}{3}u \tag{35}$$

$$\geq \quad p_i u + 1 \tag{36}$$

$$> \quad \lceil p_i u \rceil . \tag{37}$$

Note that $i \leq m$. The transition from inequality (32) to inequality (33) is due to the condition $n \geq \frac{3(m+1)^2}{\mu}$, which implies that $h \geq \frac{n}{m} - 1 \geq \frac{3(m+1)}{\mu} - 1 \geq \frac{3m}{\mu} + \frac{3}{\mu} - 1 > \frac{3m}{\mu}$. The transition from inequality (33) to inequality (34) is because of inequality (19). The transition from inequality (35) to inequality (36) is due to the setting in statement 2 in Select-Crucial-Items(.).

Therefore, with probability at most $\sum_{i=1}^{m}(P_{i,1} + P_{i,2}) \leq 2me^{-\frac{\gamma^2 u}{3}} < \alpha$, $\text{Rank}(y_i, A) \cap [ih - \mu h, ih + \mu h] = \emptyset$ for at least one $i \in \{1, \ldots, m\}$.

∎

6.2. Packing Large Items and Small Items

In this section, we show how to pack large items from sampling items in the input list. Then we show how to pack small items after packing large items.

Lemma 20. *Assume that c, η, and k are positive constants, and ϵ and δ are constants in $(0, 1)$ and θ is a constant in $[0, 1)$. Assume that the input list is S for a bin packing problem with (c, η, k)-related bins. The constants δ, μ, ϵ_1, and m are given according to equations (8) to (10). Assume that $n'_{\geq\delta}$ is an approximation of $|S_{\geq\delta}|$ satisfying*

$$(1 - \theta)|S_{\geq\delta}| \leq n'_{\geq\delta} \leq (1 + \theta)|S_{\geq\delta}|, \tag{38}$$

$$\frac{36\theta}{\delta\eta} \leq \epsilon, \text{ and} \tag{39}$$

$$\theta \left\lfloor \frac{|S_{\geq\delta}|}{m} \right\rfloor \geq 1 \quad \text{if } \theta > 0. \tag{40}$$

Let $h = \left\lfloor \frac{|S_{\geq\delta}|}{m} \right\rfloor$, $h' = \left\lfloor \frac{n'_{\geq\delta}}{m} \right\rfloor$, and S' be a list of items of size less than δ. Assume that we have the following inputs available:

- *Let $y'_1, \ldots, y'_m$ be a list of items from $S_{\geq\delta}$ such that $\text{Rank}(y'_i, S_{\geq\delta}) \cap [ih - \mu h, ih + \mu h] \neq \emptyset$ for $i = 1, 2, \ldots, m$*
- *An approximate solution for bin packing with items in $B = \{y'^{h'}_1, \ldots, y'^{h'}_m\} \cup S'$ in (c, η, k)-related bins with cost at most $(1 + \epsilon)Opt_{c,\eta,k}(B)$*

Then there Packing-Conversion(.) is an $O(1)$ time algorithm that gives an approximation app with $Opt_{c,\eta,k}(S_{\geq\delta} \cup S') \leq app \leq (1 + 5\epsilon)Opt_{c,\eta,k}(S_{\geq\delta} \cup S')$.

Proof: Assume that $a'_1 \leq a'_2 \leq \ldots \leq a'_{n_{\geq\delta}}$ is the increasing order of all input elements of size at least δ with $n_{\geq\delta} = |S_{\geq\delta}|$. Let

$$L_* = a'_1 \leq a'_2 \leq \ldots \leq a'_{n_{\geq\delta}} \cup S'. \tag{41}$$

Let

$$L_0 = a'_1 \leq a'_2 \leq \ldots \leq a'_{n'_{\geq\delta}} \cup S'. \tag{42}$$

Note that in the case $n'_{\geq\delta} > |S_{\geq\delta}|$, we let $a'_{|S_{\geq\delta}|+1} = \ldots = a'_{n'_{\geq\delta}} = 1$ in list L_0. Partition $a'_1 \leq a'_2 \leq \ldots \leq a'_{n_{\geq\delta}}$ into $A_1 y_1 A_2 y_2 \ldots A_m y_m R$ such that each A_i has exactly $h-1$ elements and R has less than h elements. Partition $a'_1 \leq a'_2 \leq \ldots \leq a'_{n'_{\geq\delta}}$ into $A_1 y_1 A_2 y_2 \ldots A_{m'} y_{m'} R'$ such that each A_i has exactly $h-1$ elements and R' has less than h elements. We have

$$\begin{aligned}
m' &= \left\lfloor \frac{n'_{\geq\delta}}{h} \right\rfloor && (43)\\
&\geq \left\lfloor \frac{(1-\theta)n_{\geq\delta}}{h} \right\rfloor && (44)\\
&\geq \frac{(1-\theta)n_{\geq\delta}}{h} - 1 && (45)\\
&\geq (1-\theta)\left\lfloor \frac{n_{\geq\delta}}{h} \right\rfloor - 1 && (46)\\
&\geq (1-2\theta)\left\lfloor \frac{n_{\geq\delta}}{h} \right\rfloor \quad \text{(by inequality (40))} && (47)\\
&= (1-2\theta)m. && (48)
\end{aligned}$$

$$\begin{aligned}
m' &= \left\lfloor \frac{n'_{\geq\delta}}{h} \right\rfloor && (49)\\
&\leq \left\lfloor \frac{(1+\theta)n_{\geq\delta}}{h} \right\rfloor && (50)\\
&\leq \frac{(1+\theta)n_{\geq\delta}}{h} + 1 && (51)\\
&\leq (1+\theta)\left\lfloor \frac{n_{\geq\delta}}{h} \right\rfloor + 1 && (52)\\
&\leq (1+2\theta)\left\lfloor \frac{n_{\geq\delta}}{h} \right\rfloor \quad \text{(by inequality (40))} && (53)\\
&= (1+2\theta)m. && (54)
\end{aligned}$$

We have

$$\begin{aligned}
h' &= \left\lfloor \frac{n'_{\geq\delta}}{m} \right\rfloor \leq \left\lfloor \frac{(1+\theta)n_{\geq\delta}}{m} \right\rfloor && (55)\\
&\leq \lfloor (1+\theta)(h+1) \rfloor && (56)\\
&\leq \lfloor (1+\theta)(h+\theta h) \rfloor \quad \text{(by inequality (40))} && (57)\\
&\leq \left\lfloor (1+\theta)^2 h \right\rfloor && (58)\\
&\leq (1+\theta)^2 h && (59)\\
&\leq (1+3\theta)h. && (60)
\end{aligned}$$

We have

$$\begin{aligned}
h' &\geq \left\lfloor \frac{(1-\theta)n_{\geq\delta}}{m} \right\rfloor \geq \left\lfloor \frac{(1-2\theta)n_{\geq\delta} + \theta n_{\geq\delta}}{m} \right\rfloor && (61)\\
&\geq \left\lfloor (1-2\theta)\frac{n_{\geq\delta}}{m} + \frac{\theta n_{\geq\delta}}{m} \right\rfloor && (62)\\
&\geq \left\lfloor (1-2\theta)\left\lfloor \frac{n_{\geq\delta}}{m} \right\rfloor + \frac{\theta n_{\geq\delta}}{m} \right\rfloor && (63)\\
&\geq \left\lfloor (1-2\theta)\left\lfloor \frac{n_{\geq\delta}}{m} \right\rfloor + 1 \right\rfloor && (64)\\
&\geq (1-2\theta)\left\lfloor \frac{n_{\geq\delta}}{m} \right\rfloor && (65)\\
&\geq (1-2\theta)h. && (66)
\end{aligned}$$

The transition from inequality (61) to inequality (66) is due to the fact $\frac{\theta n_{\geq\delta}}{m} \geq 1$ by inequalities (38) and (40). By inequalities (61) to (66), we have

$$(1+3\theta)h \geq h' \geq (1-2\theta)h. \tag{67}$$

Inequality (67) also holds if $\theta = 0$.

Consider the bin packing problems: $L_1 = y_1'^{h'} y_2'^{h'} \dots y_m'^{h'} \cup S'$. We show that there is a small difference between the results of two bin packing problems for L_0 and L_1.

Claim 20.1. For every solution of cost x with (c,η,k)-related bins for list L_0, there is a solution of cost at most $x + (10\theta + 4\mu)mh + 4h$ for list L_1.

Proof: Assume that L_0 has a bin packing solution. It can be converted into a solution for L_1 via an adaption to that of L_0 with a small number of additional bins.

We use the lots for the elements in $A_{i+1}y_{i+1}$ in L_0 to store the elements of y_i's. By inequality (67) and the assumption $\text{Rank}_\delta(y_i', S_{\geq\delta}) \cap [ih - \mu h, ih + \mu h] \neq \emptyset$ for $i = 1, 2, \dots, m$, there are at most $(3\theta + 2\mu)h$ y_i's left as unpacked for each $y_i'^{h'}$ with $i \leq m'$. Therefore, we only have that the number of elements left as unpacked in L_1 is at most

$$\begin{aligned} & m'(3\theta+2\mu)h + (|m-m'|+2)h' \\ \leq\ & (3\theta+2\mu)(1+2\theta)mh + (2\theta m+2)(1+3\theta)h \qquad \text{(by inequality (67) and (48))}. \end{aligned}$$

The number of bins for packing those left items is at most $(3\theta+2\mu)(1+2\theta)mh+(2\theta m+2)(1+3\theta)h$. Since 1 is the maximal cost of one bin, the cost for packing the left items at most

$$\begin{aligned} & (3\theta+2\mu)(1+2\theta)mh + (2\theta m+2)(1+3\theta)h \\ \leq\ & 2(3\theta+2\mu)mh + 2(2\theta m+2)h \qquad \text{(by inequality (39))} \\ \leq\ & (10\theta+4\mu)mh + 4h. \end{aligned}$$

∎

Claim 20.2. For every solution of cost y with (c,η,k)-related bins for list L_1, there is a solution of cost at most $y + (\mu + 2\theta)mh + 2h$ for list L_0.

Proof: Assume that L_1 has a bin packing solution. It can be converted into a solution for L_0 with a small number of additional bins.

We use the lots for $y_i'^{h'}$ to store the elements in A_iy_i. We have at most $(\mu + 2\theta)h$ elements left for each A_iy_i. Totally, we have at most $m(\mu + 2\theta)h + 2h$ items left. The bins for packing those left items is at most $m(\mu + 2\theta)h + 2h$, which cost at most $m(\mu + 2\theta)h + 2h$ since 1 is the maximal cost of one bin. ∎

The optimal number bins $Opt_{c,\eta,k}(L_0)$ for packing L_0 is at least $mh\delta$, which have cost at least $mh\delta\eta$. Therefore, we have

$$Opt_{c,\eta,k}(L_0) \geq mh\delta\eta \tag{68}$$

For an approximation $App(L_1)$ for packing L_1, let

$$App(L_0) = App(L_1) + (\mu + 2\theta)mh + 2h \tag{69}$$

be an approximation for packing L_0 by Claim 20.2. We have that

$$\begin{aligned} & Opt_{c,\eta,k}(L_0) && (70)\\ \leq\ & App(L_0) && (71)\\ =\ & App(L_1) + m(\mu+2\theta)h + 2h \qquad \text{(by equation (69))} && (72)\\ \leq\ & (1+\epsilon)Opt_{c,\eta,k}(L_1) + m(\mu+2\theta)h + 2h && (73) \end{aligned}$$

$$
\begin{aligned}
&\leq ((1+\epsilon)(Opt_{c,\eta,k}(L_0)+(10\theta+4\mu m)mh+4h)+m(\mu+2\theta)h+2h && (74)\\
&\quad \text{(by Claim 20.1)} && (75)\\
&\leq (1+\epsilon)(Opt_{c,\eta,k}(L_0)+Opt_{c,\eta,k}(L_0)(\frac{5\mu mh+12\theta mh+6h}{mh\delta\eta})) \quad \text{(by inequality (68))} && (76)\\
&= (1+\epsilon)Opt_{c,\eta,k}(L_0)+Opt_{c,\eta,k}(L_0)(\frac{5\mu}{\delta\eta}+\frac{12\theta}{\delta\eta}+\frac{6}{m\delta\eta}) && (77)\\
&\leq (1+\epsilon)Opt_{c,\eta,k}(L_0)+Opt_{c,\eta,k}(L_0)(\frac{5\mu}{\delta\eta}+\frac{12\theta}{\delta\eta}+\frac{\epsilon}{3}) \quad \text{(by equation (10))} && (78)\\
&\leq (1+\epsilon)(Opt_{c,\eta,k}(L_0)+Opt_{c,\eta,k}(L_0)(\frac{5\mu}{\delta\eta}+\frac{\epsilon}{3}+\frac{\epsilon}{3})) \quad \text{(by inequality (39))} && (79)\\
&\leq (1+\epsilon)(Opt_{c,\eta,k}(L_0)+Opt_{c,\eta,k}(L_0)(\frac{\epsilon}{3}+\frac{\epsilon}{3}+\frac{\epsilon}{3})) \quad \text{(by equation (8))} && (80)\\
&\leq (1+\epsilon)(1+\epsilon)Opt_{c,\eta,k}(L_0) && (81)\\
&\leq (1+3\epsilon)Opt_{c,\eta,k}(L_0). && (82)
\end{aligned}
$$

The list L_* has at most $\theta n_{\geq\delta}$ more items than L_0. Therefore

$$
\begin{aligned}
Opt_{c,\eta,k}(L_*) &= Opt_{c,\eta,k}(L_0)+\theta n_{\geq\delta} && (83)\\
&\leq Opt_{c,\eta,k}(L_0)+\frac{\theta}{1-\theta}n'_{\geq\delta} && (84)\\
&\leq Opt_{c,\eta,k}(L_0)(1+\frac{\theta n'_{\geq\delta}}{(1-\theta)Opt_{c,\eta,k}(L_0)}) && (85)\\
&\leq Opt_{c,\eta,k}(L_0)(1+\frac{\theta n'_{\geq\delta}}{(1-\theta)mh\delta\eta}) \quad \text{(by inequality (68))} && (86)\\
&\leq Opt_{c,\eta,k}(L_0)(1+\frac{\theta}{(1-\theta)h\delta\eta}\cdot\frac{n'_{\geq\delta}}{m}) && (87)\\
&\leq Opt_{c,\eta,k}(L_0)(1+\frac{\theta}{(1-\theta)h\delta\eta}\cdot 2h) \quad \text{(by inequality (17))} && (88)\\
&\leq Opt_{c,\eta,k}(L_0)(1+\frac{2\theta}{(1-\theta)\delta\eta}) && (89)\\
&\leq Opt_{c,\eta,k}(L_0)(1+\frac{4\theta}{\delta\eta}) && (90)\\
&\leq Opt_{c,\eta,k}(L_0)(1+\epsilon) \quad \text{(by inequality (39))}. && (91)
\end{aligned}
$$

Let

$$App(L_*)=(1+\epsilon)App(L_0). \tag{92}$$

Therefore, we have $App(L_*)\geq Opt_{c,\eta,k}(L_*)$ by inequality (12) and inequality (91). On the other hand, we have $App(L_*)=(1+\epsilon)App(L_0)\leq(1+\epsilon)(1+3\epsilon)Opt_{c,\eta,k}(L_0)\leq(1+\epsilon)(1+3\epsilon)Opt_{c,\eta,k}(L_*)\leq(1+5\epsilon)Opt_{c,\eta,k}(L_*)$. ∎

Algorithm Packing-Conversion$(n'_{\geq\delta}, App(L_1))$

Input: an integer $n'_{\geq\delta}$ is an approximation to $|S_{\geq\delta}|$ with $(1-\theta)|S_{\geq\delta}|\leq n'_{\geq\delta}\leq(1+\theta)|S_{\geq\delta}|$, and an approximate solution $App(L_1)$ for the bin packing with items in $L_1=\{y'^{h'}_1,\ldots,y'^{h'}_m\}\cup S'$ in (c,η,k)-related bins with cost at most $(1+\epsilon)Opt_{c,\eta,k}(L_1)$, where $L_1=\{y'^{h'}_1,\ldots,y'^{h'}_m\}\cup S'$ is a list of items such that $\text{Rank}(y'_i,S_{\geq\delta})\cap[ih-\mu h,ih+\mu h]\neq\emptyset$ for $i=1,2,\ldots,m$, and S' is a list of items of size less than δ.

Output: an approximation for $Opt_{c,\eta,k}(L_*)$, where L_* is defined by equation (41).

Steps:

Convert the approximation of $App(L_1)$ to $App(L_0)$ as equation (69) in the proof.
Convert the approximation of $App(L_0)$ to $App(L_*)$ as equation (92).
Output $App(L_*)$
End of Algorithm

Lemma 21. *Let ξ be a small constant in $(0,1)$. Assume that $S_{\geq\varphi}$ is a list of items of size at least φ, $S_{<\varphi}$ is a list of items of size less than φ, and $S'_{<\varphi}$ is another list of items of size less than φ. If $\sum_{a_i\in S_{<\varphi}} a_i + \sum_{a_i\in S_{\geq\varphi}} a_i \leq (1+\xi)(\sum_{a_i\in S'_{<\varphi}} a_i + \sum_{a_i\in S_{\geq\varphi}} a_i)$ and $\sum_{a_i\in S'_{<\varphi}} a_i + \sum_{a_i\in S_{\geq\varphi}} a_i \leq (1+\xi)(\sum_{a_i\in S_{<\varphi}} a_i + \sum_{a_i\in S_{\geq\varphi}} a_i)$, then $Opt(S_{<\varphi}\cup S_{\geq\varphi}) \leq \frac{1+\xi}{1-\varphi}\cdot Opt(S'_{<\varphi}\cup S_{\geq\varphi})+1$ and $Opt(S'_{<\varphi}\cup S_{\geq\varphi}) \leq \frac{1+\xi}{1-\varphi}\cdot Opt(S_{<\varphi}\cup S_{\geq\varphi})+1$.*

Proof: Let $L = S_{<\varphi}\cup S_{\geq\varphi}$ and $L' = S'_{<\varphi}\cup S_{\geq\varphi}$. Without loss of generality, let $Opt(L) \leq Opt(L')$. We just need to prove that $Opt(L') \leq \frac{1+\xi}{1-\varphi}\cdot Opt(L)$.

For a bin packing P for L, we convert it into another bin packing for L' by increasing small number of bins. At most one bin in P wastes more than φ space by replacing the items in $S_{<\varphi}$ with those in $S'_{<\varphi}$. If no additional bin is used for packing L', we have $Opt(L') \leq Opt(L)$.

If some new bins are needed, the total number of bins is at most

$$\begin{aligned}\frac{(\sum_{a_i\in S'_{<\varphi}} a_i + \sum_{a_i\in S_{\geq\varphi}} a_i)}{1-\varphi} + 1 &\leq \frac{(1+\xi)(\sum_{a_i\in S_{<\varphi}} a_i + \sum_{a_i\in S_{\geq\varphi}} a_i)}{1-\varphi} + 1\\ &\leq \frac{1+\xi}{1-\varphi}\cdot Opt(L) + 1.\end{aligned}$$

Therefore, we have $Opt(L') \leq \frac{1+\xi}{1-\varphi}\cdot Opt(L)$. ∎

The following Lemma 22 is only for the classical bin packing problem that all bins are of the same size 1.

Algorithm Packing-Small-Items(X, s_1, S'')
Input: $X = (x_1,\ldots,x_q)$ for the q types $T = \langle T_1,\ldots,T_q\rangle$ for the $(1+\beta)$-approximation for packing a list $S'' = \{y_1'^{h'},\ldots,y_m'^{h'}\}$, and $s_1 = \sum_{a_i\in S'} a_i$ is the sum of sizes in list S' of items of size less than δ.
Output: an approximation for $Opt(S''\cup S')$.
Steps:

1. Let $s'_1 := s_1$.
2. Repeat
3. Let $i := 1$.
4. For each type $T_i = (b_{1,i}a_1,\ldots,b_{m,i}a_m)$ (which satisfies $\sum_{j=1}^m b_{j,i}a_i \leq 1$)
5. Let $t_i := \sum_{j=1}^m b_{j,i}a_m$ and $h_i := \max(1-\delta-t_i, 0)$
6. (h_i is the available space in a bin of type T_i for packing items of size $< \delta$).
7. Let $s'_1 := \max(s'_1 - x_i h_i, 0)$ (fill each bin of type T_i with size h_i of (fractional) items).
8. Let $i := i+1$.
9. Until $s'_1 = 0$ or $i > q$.
10. If $s'_1 > 0$
11. Then find the least number k such that $k(1-\delta) \geq s'_1$

12. and fill the (fractional) items left in s'_1 into those k bins.

End of Algorithm

Lemma 22. *Let β be a constant in $(0,1)$ with $\beta \leq \frac{1}{2}$, θ be a constant in $[0,1)$ with $\theta \leq \beta$, and δ be a constant with $\delta \leq \frac{\beta}{4}$. Let m and h' be integers. Let S' be a list of items of size less than δ. Assume that $S'' = \{y_1'^{h'}, \ldots, y_m'^{h'}\}$ with $y'_i \geq \delta$ for $i = 1, \ldots, m$ and S'' is large enough to satisfy*

$$h'm \geq \frac{2}{\beta\delta}. \tag{93}$$

Then Packing-Small-Items(.) is an $O(q)$ time algorithm that given a solution $(x_1, \ldots, x_q)$ for bin packing with items in S'' with the total number of bins at most $(1+\beta)Opt(S'')$, and $s_1 = \sum_{a_i \in S'} a_i$, where x_i is the number of bins of type T_i, and q is the number of types to pack $y'_1, \ldots, y'_m$ with $q \leq m^{O(\frac{1}{\delta})}$ (see Lemma 14), it gives an approximation app for packing $S'' \cup S'$ with $Opt(S'' \cup S') \leq app \leq (1+2\beta)Opt(S'' \cup S')$.

Proof: The bin packing problem is the same as the regular bin packing problem that all bins are of the same size 1. The problem is to minimize the total number bins to pack all items. We consider the approximation to pack the small items after packing large items.

Assume that an optimal solution of a bin packing problem has two types of bins. Each first type bin contains at least one item of size δ, and each second type bin only contains items of size less than δ. Let V_1 be the set of first type bins, and V_2 be the set of all second type bins. Let U be an $(1+\beta)$-approximation for the items in S''. We have $|U| \leq (1+\beta)|V_1|$. Let $s_{large} = \sum_{a_i \in S''} a_i = \sum_{i=1}^{q} h'_i y'_i$ and $s_{small} = \sum_{a_i \in S'} a_i = s_1$.

Fill items of size less than δ into those bins in U so that each bin has less than δ left. Put all of the items less than δ into some extra bins, and at most one of them has more than δ space left. We use a fractional way to pack small items. Since each bin with small items has at least δ space left, and each small item is of size at most δ, the fractional packing of small items can be converted into a non-fractional packing. A similar argument is also shown in Lemma 21.

Case 1. If U can contain all items, we have that $|U| \leq (1+\beta)|V_1| \leq (1+\beta)|V_1 \cup V_2|$.

Case 2. There is a bin beyond those in U is used. Let U' be all bins without more than δ space left. We have

$$\begin{aligned}
|U'| &\leq \frac{s_{large} + s_{small}}{1-\delta} && (94)\\
&\leq (1 + \frac{\delta}{1-\delta})(s_{large} + s_{small}) && (95)\\
&\leq (1+2\delta)(s_{large} + s_{small}) && (96)\\
&\leq (1+\beta/2)(s_{large} + s_{small}). && (97)
\end{aligned}$$

On the other hand, $|V_1 \cup V_2| \geq s_{large} + s_{small}$. Therefore, the approximate solution $|U'| + 1$ has

$$\begin{aligned}
|U'| + 1 &\leq (1+\beta/2)|V_1 \cup V_2| + 1 \quad \text{(by inequality (97))} && (98)\\
&= (1+\beta/2)Opt(S'' \cup S') + 1 && (99)\\
&\leq (1+1.5\beta)Opt(S'' \cup S') \quad \text{(by inequality (93))} && (100)\\
&\leq (1+2\beta)Opt(S'' \cup S'). && (101)
\end{aligned}$$

Packing the items in S'' needs at least $\delta m h'$ bins. Therefore, the transition from inequality (99) to inequality (100) is by the condition in inequality (93). ∎

Algorithm Packing-With-Many-Large-Items$(\alpha, \beta, n, s_1, n'_{\geq\delta}, S)$

Input: a parameter $\beta \in (0,1)$, $n'_{\geq\delta}$ is an approximation to $|S_{\geq\delta}|$, and s_1 is an approximation for $\sum_{a_i \in S_{<\delta}} a_i$ with $(1-\xi)(\sum_{a_i \in S} a_i) \leq s_1 + \sum_{a_i \in S_{\geq\delta}} a_i \leq (1+\xi)(\sum_{a_i \in S} a_i)$, S is the list of input items $a_1, \ldots, a_n$ for bin packing, and n is the number of items in S.

Output: an approximation for $Opt(S)$.

Steps:

1. Select an integer constant d_1 such that $g_1(\frac{1}{2})^{\frac{d_1}{1-\delta}} < \alpha$.
2. Select a list L_1 of $2c_1 d_1 \frac{n}{n'_{\geq\varphi}} m^2 \log m$ random elements in the input list S,
   where constant c_1 is defined in Lemma 19, and constant d_1 is defined in line 8.
3. Let L_2 be the list of items of size at least δ in L_1.
4. Let $(y'_1, \ldots, y'_m)$:=Select-Crucial-Items(m, α, μ, L_2) (see Lemma 19).
5. Let $X = (x_1, \ldots, x_q)$:=Pack-Large-Items$(1,1,1,B)$ with $B = \{y'^{h'}_1, \ldots, y'^{h'}_m\}$
   (see Lemma 14), where $h' = \left\lfloor \frac{n'_{\geq\varphi}}{m} \right\rfloor$.
6. Let App_1:=Packing-Small-Items(X, s_1, B) (see Lemma 22).
7. Let App_2 :=Packing-Conversion$(n'_{\geq\delta}, App_1)$(see Lemma 20) for packing all items in S.
8. Output $\frac{1+\xi}{1-\delta} \cdot App_2$.

End of Algorithm

Lemma 23. *Assume that S is a list of items for bin packing problem. Let β be a constant in $(0,1)$ with $\beta \leq \frac{1}{2}$, θ be a constant in $[0,1)$ with $\theta \leq \beta$, δ be a constant with $\delta \leq \frac{\beta}{4}$, ξ be a constant with $\xi \leq \frac{\beta}{4}$, and constant $\epsilon = 6\beta$. The constants μ, ϵ_1, and m are given according to equations (8) to (10). Assume that $n'_{\geq\delta}$ is an approximation of $|S_{\geq\delta}|$ satisfying the inequalities (38), (39), (40), and (93). Assume that s_1 is an approximation for $\sum_{a_i \in S_{<\delta}} a_i$ with $(1-\xi)(\sum_{a_i \in S} a_i) \leq s_1 + \sum_{a_i \in S_{\geq\delta}} a_i \leq (1+\xi)(\sum_{a_i \in S} a_i)$. Then Packing-With-Many-Large-Items(.) is an $O(\frac{n}{\sum_{i=1}^{n} a_i} + O(\frac{1}{\beta})^{O(\frac{1}{\beta})})$ time algorithm that gives an approximation app for packing S with $Opt(S) \leq app \leq (1+16\beta)Opt(S)$ with the failure probability at most α.*

Proof: The bin packing problem is the same as the regular bin packing problem that all bins are of the same size 1. The problem is to minimize the total number bins to pack all items. We consider the approximation to pack the small items after packing large items.

We sample some random items of size at least φ from the input list S. When an item from the input list S is randomly selected, an item of size at least φ has an equal probability, which is defined by the p_φ below:

$$p_\varphi = \frac{|\{i : a_i \geq \varphi \text{ and } a_i \in \{a_1, \ldots, a_n\}\}|}{n} = \frac{n_{\geq\varphi}}{n}. \tag{102}$$

By inequality (38) and equation (102), we have

$$p_\varphi \frac{n}{n'_{\geq\varphi}} \geq \frac{1}{1+\theta}. \tag{103}$$

By Theorem 4, with probability at most $g_1(\frac{1}{2})^{p_\varphi \frac{2c_1d_1n}{n'_{\geq\varphi}} m^2 \log m} \leq g_1(\frac{1}{2})^{\frac{2c_1d_1}{1+\delta} m^2 \log m} < \alpha$ (see line 8 in Approximate-Bin-Packing(.)), we cannot obtain at least

$$\begin{aligned}
(1-\frac{1}{2})p_\varphi(\frac{2c_1d_1n}{n'_{\geq\varphi}}m^2\log m) &\geq p_\varphi \frac{n}{n'_{\geq\varphi}}(c_1d_1m^2\log m) && (104)\\
&\geq \frac{1}{1+\theta}\cdot c_1d_1m^2\log m \quad \text{(by inequality (103))} && (105)\\
&\geq c_1m^2\log m && (106)
\end{aligned}$$

random elements of size at least φ by sampling $2c_1d_1\frac{n}{n'_{\geq\varphi}}m^2\log m$ elements.

By Lemma 19, with probability at most α, we cannot obtain the list $y_1 \leq \ldots \leq y_m$ from the input list such that $\text{Rank}(y_i, S_{\geq\varphi}) \cap [ih-\mu h, ih+\mu h] \neq \emptyset$ for all $i \in \{1,\ldots,m\}$ in $O(\frac{m^2(\log m)^2)}{\mu^2})$ time using $\frac{c_1m^2\log m}{\mu^2}$ random elements from the input.

Therefore, we have probability at most $\alpha+\alpha+\alpha \leq \frac{1}{4}$, the following (a) or (b) is false:

(a). Statements 1, 2, and 3 of Lemma 9 are true.

(b). $\text{Rank}(y_i, S_{\geq\varphi}) \cap [ih-\mu h, ih+\mu h] \neq \emptyset$ for all $i \in \{1,\ldots,m\}$.

Assume that both statements (a) and (b) are true in the rest of the proof. This makes the analysis of algorithm become deterministic.

Imagine that S'_1 is a list of items of size less than δ and has $s_1 = \sum_{a_i \in S'_1} a_i$. By Lemma22, line 6 gives App_1 to be an $(1+2\beta)$-approximation for packing $S'' \cup S'_1$.

By Lemma 20, App_2 is an $(1+5\times 2\beta)$-approximation for packing $S_{\geq\delta} \cup S'_1$.

By Lemma 21, $\frac{1+\xi}{1-\delta}\cdot App_2$ is an $\frac{1+\xi}{1-\delta}\cdot(1+10\beta)$-approximation for packing $S_{\geq\delta} \cup S_{<\delta} = S$. We note that

$$\begin{aligned}
\frac{1+\xi}{1-\delta}\cdot(1+10\beta) &\leq (1+\frac{\xi+\delta}{1-\delta})\cdot(1+10\beta)\\
&\leq (1+2(\xi+\delta))\cdot(1+10\beta)\\
&\leq (1+\beta)\cdot(1+10\beta)\\
&\leq (1+\beta++10\beta+10\beta^2)\\
&\leq (1+\beta++10\beta+5\beta) \quad \text{(note that } \beta \leq \tfrac{1}{2})\\
&\leq 1+16\beta.
\end{aligned}$$

Thus, $\frac{1+\xi}{1-\delta}\cdot App_2$ is an $(1+16\beta)$-approximation for packing $S_{\geq\delta} \cup S_{<\delta} = S$.

The function is executed under the condition that $n'_{\geq\varphi} = \Omega(\sum_{i=1}^n a_i)$. Statement 2 takes $O(\frac{n}{n'_{\geq\delta}}) = O(\frac{n}{\sum_{i=1}^n a_i})$ time. The computational time at statement 5 is $(\frac{1}{\beta})^{O(\frac{1}{\beta})}$ which follows from Lemma 14. The other statements only takes $O(1)$ time. ∎

The following Lemma 24 is only for the classical bin packing problem that all bins are of the same size 1.

Algorithm Packing-With-Few-Large-Items(ξ, x, s_1)

Input: a small parameter $\xi \in [0,1)$, an integer x with $x \leq \xi\sum_{i=1}^n a_i$ and $x \geq |S_{\geq\delta}|$, and a real s_1 with $(1-\xi)(\sum_{a_i\in S} a_i) \leq s_1 + \sum_{a_i\in S_{\geq\delta}} a_i \leq (1+\xi)(\sum_{a_i\in S} a_i)$. ($s_1$ is an approximate sum of sizes of small items of size at most δ).

Output: an approximation for $Opt(S)$.

Steps:

1. Find the least number k such that $k(1-\delta) \geq s_1$
   (the k bins are for packing items of size less than δ).
2. Output $\frac{1+\xi}{1-\delta}\cdot(k+x+1)$ for packing S (x bins are for packing items of size $\geq\delta$)

End of Algorithm

Lemma 24. *Assume that S is a list of items for bin packing problem. Let δ be a constant in $(0,1)$. Assume that we have the following inputs available:*

- *x is an approximation for $|S_{\geq\delta}|$ with $x \leq \xi \sum_{i=1}^{n} a_i$ and $x \geq |S_{\geq\delta}|$ for some small $\xi \in (0,1)$.*
- *s_1 is an approximation for $\sum_{a_i \in S_{<\delta}} a_i$ with $(1-\xi)(\sum_{a_i \in S} a_i) \leq s_1 + \sum_{a_i \in S_{\geq\delta}} a_i \leq (1+\xi)(\sum_{a_i \in S} a_i)$.*

and the parameters satisfy the following conditions

$$\delta \leq \frac{1}{4}, \tag{107}$$

$$\xi \leq \frac{1}{4}, \quad \textit{and} \tag{108}$$

$$2 < \delta \sum_{i=1}^{n} a_i. \tag{109}$$

Then Packing-With-Few-Large-Items(.) is an $O(1)$ time algorithm that gives an approximation app for packing S with $Opt(S) \leq app \leq (1+8(\delta+\xi))Opt(S)$.

Proof: The bin packing problem is the same as the regular bin packing problem that all bins are of the same size 1. The problem is to minimize the total number bins to pack all items.

Imagine $S'_{<\delta}$ is a list of elements of size less than δ and $\sum_{a_i \in S'} a_i = s_1$. Let $S' = S'_{<\delta} \cup S_{\geq\delta}$. Let $s_0 = \sum_{i=1}^{n} a_i$ to be the sum of sizes of input items. By line 1 in Packing-With-Few-Large-Items(.), we have

$$\begin{aligned} k + x &\leq \frac{s_1}{1-\delta} + 1 + x & (110)\\ &\leq \frac{s_0(1+\xi)}{1-\delta} + \xi s_0 + 1 & (111)\\ &\leq (\frac{1+\xi}{1-\delta} + \xi)s_0 + 1. & (112) \end{aligned}$$

Furthermore, assume that the inequalities (107) to (109) holds. We have

$$\begin{aligned} (\frac{1+\xi}{1-\delta} + \xi) &\leq (1 + \frac{\xi+\delta}{1-\delta} + \xi)\\ &\leq (1 + 2(\xi+\delta) + \xi)\\ &= (1 + 2\delta + 3\xi) \end{aligned}$$

Therefore, we have

$$\begin{aligned} k + x + 1 &\leq (1 + 2\delta + 3\xi)Opt(S) + 1\\ &\leq (1 + 3\delta + 3\xi)Opt(S) \qquad \text{(by inequality (109))}. \end{aligned}$$

By Lemma 21, we have the $\frac{1+\xi}{1-\delta}(1+3\delta+3\xi)$-approximation for packing S. We note that

$$\begin{aligned} \frac{1+\xi}{1-\delta}(1+3\delta+3\xi) &\leq (1 + \frac{\xi+\delta}{1-\delta})(1+3\delta+3\xi)\\ &\leq (1 + 2(\xi+\delta))(1+3\delta+3\xi)\\ &\leq 1 + 2(\xi+\delta) + (3\delta+3\xi) + 2(\xi+\delta)(3\delta+3\xi)\\ &\leq 1 + 2(\xi+\delta) + 3(\delta+\xi) + 3(\delta+\xi)\\ &\leq 1 + 8(\xi+\delta). \end{aligned}$$

∎

6.3. Full Sublinear Time Approximation Scheme for Bin Packing

Now we present a sublinear time approximation scheme for the bin packing problem. The brief idea of our sublinear time algorithm is given in Section 2.1. After setting up some parameters, it divides the interval $(0, 1]$ for item sizes into $O(\log n)$ intervals $(0, 1] = I_1 \cup \ldots \cup I_k$, called a $(\varphi, \delta, \gamma)$-partition as described in section 3. Applying the algorithm described in section 3, we get an approximation about the distribution of the items in the intervals $I_1, \ldots, I_k$. If the total size $\sum_{i=1}^{n} a_i$ is too small, for example $O(1)$, the linear time algorithm described in section 5 is used to output an approximation for the bin packing problem. Otherwise, we give a sublinear time approximation for the bin packing problem. In order to pack large items, we derive the approximate crucial items, which are the approximate ih-th elements among the large items of size at least φ for $i = 1, \ldots, m$, where h and m are defined in equations (16), and ((10)), respectively. The algorithm described in section 5 is used to pack large items. The small items are filled into bins which have space left after packing large items, and some additional fresh bins. With the approximate sum of sizes of small items, we can calculate the approximate number of fresh bins to be needed to pack them. If the total sum of the sizes of large items is too small to affect the total approximation ratio, we just directly pack the small items according to approximate sum of the sizes of those small objects.

Algorithm Approximate-Bin-Packing(τ, n, S)
Input: a positive real number τ, an integer n, and a list S of n items $a_1, \ldots, a_n$ in $(0, 1]$.
Output: an approximation *app* with $Opt(S) \leq app \leq (1 + \tau)Opt(S) + 1$.
Steps:

1. Let $\beta := \frac{\tau}{30}$ and $\epsilon := 6\beta$.
2. Let $\delta := \frac{\epsilon}{4}$ and $\theta := \frac{\epsilon\delta}{36}$.
3. Let μ, ϵ_1 and m are selected by equations (8), (9), and (10), respectively.
4. Let $c := \eta := k := 1$ (classical bin packing).
5. Let $\alpha := 1/12$.
6. Let $\varphi := \delta$.
7. Let $\gamma := \delta^3$.
8. Select an integer constant d_1 such that $g_1(\frac{1}{2})^{\frac{d_1}{1-\delta}} < \alpha$.
9. Derive a $(\varphi, \delta, \gamma)$-partition $P = I_1 \cup \ldots \cup I_k$ for $(0, 1]$.
10. Let $(s, s_1, n'_{\geq\varphi})$:=Approximate-Interval$(\varphi, \delta, \gamma, \theta, \alpha, P, n, S)$ (see Lemma 9).
11. If $s < \max((\frac{4m}{\theta\delta^2}), (\frac{4}{\delta^2} \cdot \frac{(1+\theta)m}{\theta}), (\frac{16}{\delta^2} \cdot \frac{(1+\theta)}{\beta\delta}))$
12. then
13. Output Linear-Time-Packing(n, S) (see Lemma 16) and terminate the algorithm.
14. If $n'_{\geq\varphi} \geq \frac{\delta^2}{4}s$
15. then
16. Output Packing-With-Many-Large-Items$(\alpha, \beta, n, s_1, n'_{\geq\delta}, S)$ (see Lemma 23).
17. else
18. If $n'_{\geq\varphi} > 0$

19. then let $x := \frac{n'_{\geq\varphi}}{1-\theta}$ and $\xi := \max(\delta^2, \theta + \delta^3)$

20. else let $x := 6\delta s$ and $\xi := \max(12\delta, \theta + \delta^3)$.

21. Output Packing-With-Few-Large-Items(ξ, x, s_1) (see Lemma 24).

End of Algorithm

Proof: [Theorem 10] Calling function Approximate-Interval(.) in line 10 in the algorithm Approximate-Bin-Packing(.), we obtain s for an approximate sum $\sum_{i=1}^{n} a_i$ of items in list S, s_1 for an approximate sum of items in list $S_{<\varphi}$, an approximate number $n'_{\geq\varphi}$ of items of size at least φ (see Lemma 9). With probability at most α, at least one of statements 1, 2, 3, 4, and 5 of Lemma 9 of Lemma 9 is false. Therefore, we have probability at most α, the following statement (a) is false:

(a). Statements 1, 2, 3, 4, and 5 of Lemma 9 are true.

Assume that statement (a) is true in the rest of the proof. By statement 1 of Lemma 9, we have that if $n'_{\geq\varphi} > 0$, then

$$(1-\theta)n_{\geq\varphi} \leq n'_{\geq\varphi} \leq (1+\theta)n_{\geq\varphi}. \tag{113}$$

Let $s_0 = \sum_{i=1}^{n} a_i$. By line 10 in Approximate-Bin-Packing(.) and Lemma 9, s is an approximation of $s_0 = \sum_{i=1}^{n} a_i$, s_1 is an approximation of $\sum_{i=1, a_i<\varphi}^{n} a_i$, and $n'_{\geq\varphi}$ is an approximation of the number $n_{\geq\varphi}$ of items of size at least φ. A $(\varphi, \delta, \gamma)$-partition for $(0, 1]$ divides the interval $(0, 1]$ into intervals $I_1 = [\pi_1, \pi_0], I_2 = (\pi_2, \pi_1], I_3 = (\pi_3, \pi_2], \ldots, I_k = (0, \pi_{k-1}]$ as in Definition 7.

Claim 10.1. If the condition in line 11 of Approximate-Bin-Packing(.) is true, the algorithm outputs an approximation $app(S)$ for the bin packing problem S with $Opt(S) \leq app(S) \leq (1+\tau)Opt(S) + 1$.

Proof: We note that if the condition in line 11 is true, then $s = O(1)$ since β, θ, m, and δ are all constants. By statement 3 of Lemma 9, we have $s_0 = O(1)$. In this case, we use the linear time deterministic algorithm by Lemma 16, which warrants the desired ratio of approximation. ∎

In the rest of the proof, we assume that the condition in line 11 is false. We have the inequality:

$$s \geq \max((\frac{4m}{\theta\delta^2}), (\frac{4}{\delta^2} \cdot \frac{(1+\theta)m}{\theta}), (\frac{16}{\delta^2} \cdot \frac{(1+\theta)}{\beta\delta})). \tag{114}$$

By inequality (114), we have the inequality

$$s \geq \frac{8}{\delta^2} \cdot \frac{1}{\beta\delta} \geq \frac{8}{\delta^3}. \tag{115}$$

Therefore,

$$\delta \leq \frac{\delta^4 s}{8}. \tag{116}$$

By statement 3 of Lemma 9, we have $s \leq (1+\theta)(\sum_{i=1}^{n} a_i) = (1+\theta)s_0$. By inequality (115) and the fact $\delta \leq 1$ (by the setting in line 2), we have

$$s_0 \geq \frac{s}{1+\theta} \geq \frac{s}{2} \geq \frac{4}{\delta^3} \geq 4. \tag{117}$$

By inequality (117) and statement 4 of Lemma 9, we have

$$\frac{1}{4}(1-\theta)(1-\delta)\varphi(\sum_{i=1}^{n} a_i) \leq s \leq (1+\theta)s_0. \tag{118}$$

Claim 10.2. If the condition at line 14 of the algorithm Approximate-Bin-Packing(.) is true, then with failure probability at most α, the algorithm outputs an approximation $app(S)$ for the bin packing problem with $Opt(S) \leq app(S) \leq (1+\tau)Opt(S) + 1$.

Proof: Assume that the condition at line 14 of the algorithm Approximate-Bin-Packing(.) is true. This is the case that the number of large items is large. The condition of line 11 in Approximate-Bin-Packing(.) is false. Since condition of line 14 in Approximate-Bin-Packing(.) is true, we have

$$\begin{aligned}
h'm &\geq \left\lfloor \frac{n'_{\geq\varphi}}{m} \right\rfloor m && (119)\\
&\geq (\frac{n'_{\geq\varphi}}{m} - 1)m && (120)\\
&= n'_{\geq\varphi} - m && (121)\\
&\geq \frac{\delta^2}{4}s - m && (122)\\
&\geq \frac{\delta^2}{4}s - \frac{s}{64} \quad \text{(by inequality (114))} && (123)\\
&\geq \frac{\delta^2}{8}s && (124)\\
&\geq \frac{2}{\beta\delta}, \quad \text{(by inequality (114))} && (125)
\end{aligned}$$

where h' is defined is statement 5 of Packing-With-Many-Large-Items(.). Note that the transition from inequality (121) to inequality (122) is due to condition of line 14 in Approximate-Bin-Packing(.) is true, and the transition from inequality (122) to inequality (125) is due to inequality (114), Thus, the inequality (93) condition in Lemma 20 is true.

Inequality (38) is satisfied because of inequality (113). Inequality (39) is satisfied because of the setting in lines 1 to 4 of Approximate-Bin-Packing(.). We have the inequality

$$\begin{aligned}
\theta \left\lfloor \frac{n_{\geq\varphi}}{m} \right\rfloor &\geq \theta \left\lfloor \frac{n'_{\geq\varphi}}{(1+\theta)m} \right\rfloor && (126)\\
&\geq \theta \left\lfloor \frac{\delta^2 s}{4(1+\theta)m} \right\rfloor && (127)\\
&\geq \theta \left\lfloor \frac{\delta^2 \cdot (\frac{4}{\delta^2} \cdot \frac{(1+\theta)m}{\theta})}{4(1+\theta)m} \right\rfloor && (128)\\
&\geq \theta \left\lfloor \frac{(1+\theta)}{\theta} \right\rfloor && (129)\\
&\geq \theta \left\lfloor \frac{1}{\theta} + 1 \right\rfloor && (130)\\
&\geq \theta \cdot \frac{1}{\theta} = 1. && (131)
\end{aligned}$$

The transition from inequality (126) to inequality (127) is because the condition of statement 14 of Approximate-Bin-Packing(.)) is true. The transition from inequality (127) to inequality (128) is because of inequality (114). Thus, inequality (40) is satisfied.

By Lemma 23, the algorithm gives an approximation $app(S)$ such that $Opt(S) \leq app(S) \leq (1+16\beta)Opt(S) \leq (1+\tau)Opt(S)$ (by the setting of β in statement 1 of Approximate-Bin-Packing(.)) with the failure probability at most α. ∎

Claim 10.3. If the condition at line 14 of the algorithm Approximate-Bin-Packing(.) is false, then the algorithm outputs an approximation $app(S)$ for the bin packing problem with $Opt(S) \leq app(S) \leq (1+\tau)Opt(S) + 1$.

Proof: In the case that the condition at line 14 does not hold, we have that

$$n'_{\geq\varphi} < \frac{\delta^2}{4}s \quad (132)$$

$$\leq \frac{\delta^2}{4}(1+\delta)s_0 \quad \text{(by inequality (118))} \tag{133}$$

$$\leq \frac{\delta^2}{2}s_0. \tag{134}$$

Line 21 in the algorithm Approximate-Bin-Packing(.) will be executed. By inequality (117), inequality (109) is true. Inequalities (107) and (108) follow from lines 1 and 2 in the Algorithm Approximate-Bin-Packing(.).

By statements 1 and 2 of Lemma 9, we have

$$s_1 = \sum_{\hat{C}(I_j,S)>0 \text{ and } j>1} \hat{C}(I_j,S)\pi_j \tag{135}$$

$$\geq \sum_{\hat{C}(I_j,S)>0 \text{ and } j>1} (1-\theta)C(I_j,S)\pi_j \quad \text{(by statement 1 of Lemma 9)} \tag{136}$$

$$\geq (1-\theta) \sum_{a_i\in I_j \text{ with } \hat{C}(I_j,S)>0 \text{ and } j>1} a_i \tag{137}$$

$$\geq (1-\theta) \sum_{a_i\in I_j \text{ and } j>1} a_i - \sum_{a_i\in I_j \text{ with } \hat{C}(I_j,S)=0 \text{ and } j>1} a_i \tag{138}$$

$$\geq (1-\theta) \sum_{a_i\in S_{<\varphi}} a_i - (\frac{\delta^3}{2}\sum_{a_i\in S_{<\varphi}} a_i + \frac{\gamma}{n}). \quad \text{(by statement 2 of Lemma 9)} \tag{139}$$

$$\tag{140}$$

We have

$$s_1 + \sum_{a_i\in S_{\geq\varphi}} a_i \geq (1-\theta)(\sum_{a_i\in S_{<\varphi}} a_i) - (\frac{\delta^3}{2}\sum_{a_i\in S_{<\varphi}} a_i + \frac{\gamma}{n}) + \sum_{a_i\in S_{\geq\varphi}} a_i \tag{141}$$

$$\geq (1-\theta)(\sum_{a_i\in S} a_i) - (\frac{\delta^3}{2}\sum_{a_i\in S_{<\varphi}} a_i + \frac{\gamma}{n}) \tag{142}$$

$$\geq (1-\theta)(\sum_{a_i\in S} a_i) - (\frac{\delta^3}{2}\sum_{a_i\in S} a_i + \frac{\gamma}{n}) \quad \text{(note } S_{<\varphi}\subseteq S\text{)} \tag{143}$$

$$\geq (1-\theta-\frac{\delta^3}{2})(\sum_{a_i\in S} a_i) - \frac{\gamma}{n} \tag{144}$$

$$\geq (1-\theta-\delta^3)(\sum_{a_i\in S} a_i). \quad \text{(by inequality (117))} \tag{145}$$

By statements 1 and 2 of Lemma 9, we have

$$s_1 = \sum_{\hat{C}(I_j,S)>0 \text{ and } j>1} \hat{C}(I_j,S)\pi_j \tag{146}$$

$$\leq \sum_{\hat{C}(I_j,S)>0 \text{ and } j>1} (1+\theta)C(I_j,S)\pi_j \quad \text{(by statement 1 of Lemma 9)} \tag{147}$$

$$\leq \frac{1+\theta}{1-\varphi} \sum_{a_i\in I_j \text{ with } \hat{C}(I_j,S)>0 \text{ and } j>1} a_i \tag{148}$$

$$\leq \frac{1+\theta}{1-\varphi} \sum_{a_i\in S_{<\varphi}} a_i. \tag{149}$$

By statement 1 of Lemma 9, we have

$$\begin{aligned} s_1 + \sum_{a_i \in S_{\geq\varphi}} a_i &\leq \frac{1+\theta}{1-\varphi} \sum_{a_i \in S} a_i && (150)\\ &\leq (1+\theta)(1+2\varphi) \sum_{a_i \in S} a_i && (151)\\ &\leq (1+\theta+4\varphi) \sum_{a_i \in S} a_i. && (152) \end{aligned}$$

Therefore,

$$(1-(\theta+\delta^3))(\sum_{a_i \in S} a_i) \leq s_1 + \sum_{a_i \in S_{\geq\varphi}} a_i \leq (1+(\theta+4\varphi))(\sum_{a_i \in S} a_i). \quad (153)$$

Since the condition at line 14 in Approximate-Bin-Packing(.) is false, we discuss two cases

- Case $n'_{\geq\varphi} > 0$.
  We have the inequalities

$$\begin{aligned} \sum_{a_i \geq \varphi} a_i &\leq n_{\geq\varphi} && (154)\\ &\leq (1+\theta) n'_{\geq\varphi} \quad \text{(by inequality (113))} && (155)\\ &\leq 2n'_{\geq\varphi} && (156)\\ &\leq \delta^2 s_0. \quad \text{(by inequality (134))} && (157) \end{aligned}$$

  By statement 1 of Lemma 9, we have

$$\frac{n'_{\geq\varphi}}{1-\theta} \geq |S_{\geq\delta}|. \quad (158)$$

  We also have

$$\begin{aligned} \frac{n'_{\geq\varphi}}{1-\theta} &\leq \frac{\delta^2}{4(1-\theta)} s \quad \text{(line 14 in Approximate-Bin-Packing(.) is false)} && (159)\\ &\leq \frac{\delta^2}{2} s && (160)\\ &\leq \frac{\delta^2}{2}(1+\delta) s_0 \quad \text{(by inequality (118))} && (161)\\ &\leq \delta^2 s_0. && (162) \end{aligned}$$

  In this case, $x = \frac{n'_{\geq\varphi}}{1-\theta}$ by inequality (158) and inequalities (159) to (162), and $\xi = \max(\delta^2, \theta + \delta^3)$ by inequality (153). They satisfy the conditions of Lemma 24, which implies that the approximation ratio is $(1+8(\delta+\xi)) \leq (1+\tau)$ by the assignments in lines 1 and 2 in algorithm Approximate-Bin-Packing(.).

- Case $n'_{\geq\varphi} = 0$.
  By statement 2 of Lemma 9, we have

$$\begin{aligned} \delta|S_{\geq\varphi}| &\leq \sum_{a_i \geq \varphi} a_i \\ &= \sum_{a_i \in I_1} a_i \end{aligned}$$

$$
\begin{aligned}
&\leq \frac{\delta^3}{2}s_0 + \frac{\gamma}{n} \quad \text{(apply statement 2 of Lemma 9 with } \hat{C}(I_1, S) = n'_{\geq\varphi} = 0) \\
&\leq \frac{\delta^3}{2}s_0 + \delta \\
&\leq \frac{\delta^3}{2}s_0 + \frac{\delta^4 s}{8} \quad \text{(by inequality (116))} \\
&\leq \frac{\delta^3}{2}s_0 + \frac{\delta^4(1+\delta)s_0}{8} \quad \text{(by inequality (118))} \\
&\leq \frac{\delta^3}{2}s_0 + \frac{\delta^4 s_0}{4} \\
&\leq \frac{3\delta^3}{4}s_0 \\
&\leq \frac{3\delta^3}{4}\frac{8s}{\delta} \quad \text{(by inequality (118))} \\
&\leq 6\delta^2 s.
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
|S_{\geq\varphi}| &\leq 6\delta s && (163)\\
&\leq 6\delta(1+\delta)s_0 && (164)\\
&\leq 12\delta s_0. && (165)
\end{aligned}
$$

In this case, let $x = 6\delta s$ by inequality (163), and let $\xi = \max(12\delta, \theta + \delta^3, 1 + \theta + 4\varphi)$ by inequality (153) and inequalities (163) to (165). They satisfy the conditions of Lemma 24, which implies the approximation ratio is $(1 + 8(\delta + \xi)) \leq (1 + \tau)$ by the assignments in lines 1 and 2 in algorithm Approximate-Bin-Packing(.). This completes the proof of Claim 10.3. ∎

Claim 10.4. The algorithm runs in $O(\frac{n(\log n)(\log\log n)}{\sum_{i=1} a_i} + (\frac{1}{\tau})^{O(\frac{1}{\tau})})$ time.

Proof: We give the computational time about the algorithm. Lines 1 to 8 take $O(1)$ time. Line 9 takes $O(\log n)$ time. By Lemma 9, Line 10 takes $O((\frac{n}{\sum_{i=1}^{n} a_i})(\log n)\log\log n))$ time. Line 13 takes $O(n)$ time by calling Linear-Time-Packing(S) by Lemma 16. This only happens when $\sum_{i=1}^{n} a_i = O(1)$.

By Lemma 23, statement 16 of Approximate-Bin-Packing(.) takes $O(\frac{n}{\sum_{i=1}^{n} a_i} + O(\frac{1}{\beta})^{O(\frac{1}{\beta})}) = O(\frac{n}{\sum_{i=1}^{n} a_i} + O(\frac{1}{\tau})^{O(\frac{1}{\tau})})$ time.

Line 21 takes $O(1)$ time by Lemma 24. Therefore, in the worst case, the algorithm takes $O(\frac{n(\log n)(\log\log n)}{\sum_{i=1} a_i} + (\frac{1}{\tau})^{O(\frac{1}{\tau})})$ time.

Claim 10.5. The failure probability of the algorithm is at most $\frac{1}{4}$.

Proof: Two statements 10 and 16 in the algorithm may fail due to randomization. Each of them has probability at most α to fail by Lemma 9 (for statement (a)), and Claim 10.2. Therefore, the failure probability of the entire algorithm is at most $2\alpha \leq \frac{1}{4}$. ∎

∎

The theorem follows from the above claims. This completes the proof of Theorem 10

∎

The following Theorem 25 gives a dense sublinear time hierarchy approximation scheme for bin packing problem.

Theorem 25. *For each $\epsilon \in (0, 1)$, and $b \in (0, 1]$, there is a randomize $(1 + \epsilon)$-approximation for all $\sum(n^b)$-bin packing problems in time $O(n^{1-b}(\log n)\log\log n)$ time, but there is no $o(n^{1-b})$ time $(1 + \epsilon)$-approximation algorithm $\sum(n^b)$-bin packing problem.*

Proof: It follows from Theorem 10 and Theorem 11. ∎

6.4. NP Hardness

In this section, we show that $\sum(n^b)$ and $S(\delta)$ are both NP-hard. We reduce the 3-partition problem, which is defined below, to them.

Definition 26. The *3-partition problem* is to decide whether a given multiset of integers in the range $(\frac{B}{4}, \frac{B}{2})$ can be partitioned into triples that all have the same sum B, where B is an integer. More precisely, given a multiset S of $n = 3t$ positive integers, can S be partitioned into m subsets $S_1, S_2, \ldots, S_t$ such that the sum of the numbers in each subset is equal?

It is well known that 3-partition problem is NP-complete [14]. It is used in proving the following NP-hard problems (Theorem 27 and Theorem 28)

Theorem 27. *For each constant $b \in (0,1)$, the bin packing problem in $\sum(n^b)$ is NP-hard.*

Proof: We construct a reduction from 3-partition problem to the $\sum(n^b)$-bin packing problem via some padding. Assume that $b_1, \ldots, b_n$ is a list of 3-partition problem with all items in $(\frac{B}{4}, \frac{B}{2})$. The bin packing problem for $\sum(n^b)$ is constructed below:

It has a new list of elements: $a_1, \ldots, a_n, a_{n+1}, \ldots, a_m$ such that $\sum_{i=1}^{m} a_i = m^b$, where $a_i = \frac{b_i}{B}$ for $i = 1, \ldots, n$, and each a_j with $j > n$ is 1, $1 - \frac{1}{5}$ or in $(0, \frac{1}{5}]$. Furthermore, there are at most five items of size $1 - \frac{1}{5}$. Let $m = \left\lceil n^{\frac{2}{b}} \right\rceil$. Therefore, we have $m^b \geq n^2$. This makes us the sufficient flexibility to select those items a_i with $i > n$. Let $s = \sum_{i=1}^{n} a_i$. Select a number n_1 such that $m^b - 5 \leq (n_1 - n) + s < m^b - 4$. In other words, we have $m^b + n - s - 5 \leq n_1 < m^b + n - s - 4$. Thus, for all large n, we also have $n_1 < m^b + n - s - 4 \leq m^b + n \leq 2m^b < \frac{m}{2}$ since $b < 1$. Let $a_i = 1$ for all $i = n+1, \ldots, n_1$. Therefore, $\sum_{i=1}^{n_1} a_i \in [m^b - 5, m^b - 4)$. Then we select a_i with $i = 1, \ldots, m$ so that $\sum_{i=1}^{m} a_i = m^b$. We select the next five items $a_{n'+1}, \ldots, a_{n'+5}$ of size $1 - \frac{1}{5}$. Thus, $\sum_{i=1}^{n_1} a_i \in [m^b - 1, m^b)$. Let $r = m^b - \sum_{i=1}^{n_1} a_i$. We have $r \in (0, 1]$. The rest items $a_{n'+6}, a_{n'+7}, \ldots, a_m$ are partitioned into five groups G_1, G_2, G_3, G_4, and G_5 that size difference between any two of them is at most one. Each item in G_i is assigned $\frac{r}{5|G_i|} \in (0, \frac{1}{5}]$. Thus, 1) $\sum_{a_i \in G_j} a_i = \frac{r}{5}$; 2)$\sum_{i=n'+6}^{m} a_i = r$; and 3) $\sum_{i=1}^{m} a_i = m^b$.

There is an optimal bin packing solution such that the five items of size $1 - \frac{1}{5}$ are in five bins with all items in the range $(0, \frac{1}{5}]$. There is a solution for the 3-partition problem if and only if the bin packing problem can be solved with $\frac{n}{3} + (n_1 - n) + 5$ bins. Any packing with $\frac{n}{3} + (n_1 - n) + 5$ bins for $a_1, a_2, \ldots, a_m$ has to be the case that each item a_j with $j > n' + 5$ is in a bin containing one item of size $1 - \frac{1}{5}$ since it is impossible for a_i $(i \leq n)$ to share a bin with a_j $(n' + 1 \leq j \leq n' + 5)$. ∎

Combining Theorem 27 and Theorem 25, we see a sublinear time hierarchy of approximation scheme for a class of NP-hard problems, which are derived from bin packing problem. We show that the $S(\delta)$-bin packing problem is NP-hard if δ is at least $\frac{1}{4}$.

Theorem 28. *For each δ at most $\frac{1}{4}$, the $S(\delta)$-bin packing problem is NP-hard.*

Proof: We reduce the 3-partition problem to $S(\delta)$-bin packing problem. Assume that $S = \{a_1, \ldots, a_{3m}\}$ is an input of 3-partition. We design that a $S(\delta)$-bin packing problem as below: the bin size is 1 and the items are $\frac{a_1}{B}, \ldots, \frac{a_{3m}}{B}$. The size of each item is at least $\frac{1}{4}$ since each $a_i > \frac{B}{4}$. It is easy to see that there is a solution for the 3-partition problem if and only if those items for the bin packing problem can be packed into m bins. ∎

7. Constant Time Approximation Scheme

In this section, we show that there is a constant time approximation for the $S(\delta)$-bin packing problem with (c, η, k)-related bins for any positive constant δ.

Lemma 29. *Assume that c, η, and k are constants. Assume there is a $t(m, n, \mu)$ time and $z(m, n, \mu)$ queries algorithm A such that given a list S of items of size at least δ, it returns m items $y'_1, y'_2, \ldots, y'_m$ from the list with $\mathrm{Rank}(y'_i, S) \cap [ih - \mu h, ih + \mu h] \neq \emptyset$ for $i = 1, 2, \ldots, m$. Then there is an $z(m, n, \mu)$ queries and $t(m, n, \mu) + (\frac{1}{\epsilon\delta})^{O(\frac{1}{\delta})}$ time approximation scheme B for the $S(\delta)$-bin packing problem with (c, η, k)-related bins. Furthermore, if A fails with probability at most α, then B also fails with probability α.*

Proof: Assume that c, η, and k are positive constants. Let ϵ be an arbitrary positive constant. The constants μ, ϵ_1, and m are given according to equations (8) to (10). We let the number of elements n be large enough such that $\frac{2q}{n\delta\eta} < \frac{\epsilon}{3}$, where q is defined at Lemma 14.

Assume that $a'_1 \leq a'_2 \leq \ldots \leq a'_{n_{\geq\delta}}$ is the increasing order of all input elements at least δ. Let $L_0 = a'_1 \leq a'_2 \leq \ldots \leq a'_n$. We partition them into $y_0 A_1 y_1 A_2 y_2 \ldots A_m y_m R$ such that each A_i has exactly h elements and R has less than h elements.

Using algorithm A, we make approximation y'_i to y_i such that the rank of y'_i has at most μh distance with that of y_i. Assume that $\mathrm{Rank}(y'_i, S) \cap [ih - \mu h, ih + \mu h] \neq \emptyset$ for $i = 1, 2, \ldots, m$ from algorithm A.

By Lemma 14, we have approximation scheme for $\{y'^h_1, \ldots, y'^h_m\}$ with computational time $(\frac{1}{\epsilon\delta})^{O(\frac{1}{\delta})}$, which follows from Lemma 14 and the selection of m and μ. The approximation scheme for $S(\delta)$-bin packing problem follows from Lemma 20. The total time is $t(m, n, \mu) + (\frac{1}{\epsilon\delta})^{O(\frac{1}{\delta})}$ for running A and time involved in the algorithm of Lemma 14. ∎

Lemma 29 is applied in both deterministic and randomized algorithms in this paper. We note that algorithm A in Lemma 29 is deterministic if $\alpha = 0$.

For the bin packing problem with item of size at least a positive constant, our Theorem 30 generalizes a result in [3].

Theorem 30. *Assume that c, η, and k are constants. There is an $O(\frac{1}{\delta^2\epsilon^4})$ queries and $(\frac{1}{\epsilon\delta})^{O(\frac{1}{\delta})}$ time randomized approximation scheme algorithm for the $S(\delta)$-bin packing problem with (c, η, k)-related bins.*

Proof: Let S be the list of input items of size at least δ. It follows from Lemma 19, and Lemma 29. By Lemma 19, we have a $t(m, n, \mu) = O(\frac{m^2(\log m)^2)}{\mu^2})$ time algorithm such that using $z(m, n, \mu) = O(\frac{m^2 \log m}{\mu^2})$ random elements from A, it generates elements $y'_1 \leq \ldots \leq y'_m$ from the input list such that $\Pr[\mathrm{Rank}(y'_i, S) \cap [ih - \mu h, ih + \mu h] = \emptyset$ for at least one $i \in \{1, \ldots, m\}] \leq \alpha$. We assume that the m items $y'_1, y'_2, \ldots, y'_m$ satisfy $\mathrm{Rank}(y'_i, S) \cap [ih - \mu h, ih + \mu h] \neq \emptyset$ for $i = 1, 2, \ldots, m$. The approximation scheme follows from Lemma 29. ∎

Corollary 31 ([3]). *There is an $O(\frac{1}{\delta^2\epsilon^4})$ queries and $(\frac{1}{\epsilon\delta})^{O(\frac{1}{\delta})}$ time approximation scheme algorithm for the $S(\delta)$-bin packing problem.*

We have Theorem 32 that shows an example of NP-hard problem that has a constant time approximation scheme.

Theorem 32. *There is an NP-hard problem that has a constant time approximation scheme.*

Proof: It follows from Theorem 28 and Corollary 31. ∎

8. Streaming Approximation Scheme

In this section, we show a constant time and constant space streaming algorithm for the bin packing problem. For the streaming model of the bin packing problem, we output a plan to pack the items that have come from the input list, and the number of bins to approximate the optimal number of bins. Our algorithm only holds a constant number of items. Therefore, it has a constant updating time and constant space complexity.

Lemma 33. *There is an $O(u)$ updating time algorithm to select u random elements from a stream of input elements.*

Proof: We set up u positions to put the u elements. There is a counter n to count the total number of elements arrived. For each new arrived element a_n, the j-th position uses probability $\frac{1}{n}$ to replace the old element at the j-th position with the new element. For each element a_i, with probability $\frac{1}{j}\frac{j}{j+1}\ldots\frac{n-1}{n} = \frac{1}{n}$, it is kept at each of the u positions after processing n elements. Therefore, we keep u-random elements from the input list. ∎

A brief description of our streaming algorithm for the bin packing problem is given in section 2.1. Using the method of Lemma 33, we maintain a list X of $O(1)$ random items of large sizes from the input list. The list is updated after receiving every new element. The sizes of each small item is added into a variable s_1. Using the method in section 6.1, we find the approximate crucial items from the list X of random large items, which are the approximate ih-th elements among the large items of size at least δ for $i = 1, \ldots, m$, where h and m are defined in equations (16), and ((10)), respectively. The algorithm described in section 5 is used to pack large items. The small items are filled into bins which have space left after packing large items, and some additional fresh bins. With the sum s_1 of sizes of small items, we can calculate the approximate number of fresh bins to be needed to pack them.

Algorithm Streaming-Bin-Packing
Input: a positive constant ϵ, and a streaming of items of size at least δ.
Output: an $(1+\epsilon)$-approximation.
Steps:

1. Let $\beta := \frac{\gamma}{30}$ and $\epsilon := 6\beta$.
2. Let $\delta := \frac{\epsilon}{4}$ and $\theta := 0$.
3. Let μ, ϵ_1 and m are selected by equations (8), (9), and (10), respectively.
4. Let $c := \eta := k := 1$ (classical bin packing).
5. Let $\alpha := 1/8$.
6. Let $u := \frac{c_1 m^2 \log m}{\mu^2}$, where c_1 is defined in Lemma 19.
7. Let $v := \frac{2m}{\beta\delta} + m$.
8. Let $X[1...u]$ be an array of u elements.
9. Let $X[i] := 0$ for $i = 1, \ldots, u$.
10. Let $Y[1...v]$ be an array of v elements.
11. Let $Y[i] := 0$ for $i = 1, \ldots, v$.
12. Let $n := 0$.
13. Let $n_{\geq\delta} := 0$.

14. Let $s_1 := 0$.

15. For each new element a_i

16. Let $n := n + 1$.

17. If $a_i < \delta$

18. then

19. Let $s_1 := s_1 + a_i$.

20. else

21. Let $n_{\geq\delta} := n_{\geq\delta} + 1$.
If $n_{\geq\delta} < v$ then let $Y[n_{\geq\delta}] := a_i$.

22. For $i = 1$ to u, let each $X[i]$ take the new elements with probability $\frac{1}{n_{\geq\delta}}$.

23. If $n_{\geq\delta} > v$

24. then

25. Output Packing-With-Many-Large-Items$(\alpha, \beta, n, s_1, n_{\geq\delta}, S)$ (see Lemma 23).

26. else

27. Let $(b_1, \ldots, b_t)$ =Linear-Time-Packing$(n_{\geq\varphi}, Y)$ (see Lemma 16) (each bin b_i represents a packing of items in Y).

28. For each b_i with left space $u_i > \delta$,

29. move $u_i - \delta$ (fractional)item size into b_i from s_1, and let $s_1 = s_1 - (u_i - \delta)$.

30. Allocate s_1 into fresh bins such that each bin except the last one wastes δ space.

End of Algorithm

Theorem 34. *Streaming-Bin-Packing is a single pass randomized streaming approximation scheme for the bin packing problem such that it has $O(1)$ updating time and $O(1)$ space, and computes an approximate packing solution $Apx(n)$ with $\mathrm{Sopt}(n) \leq App(n) \leq (1+\epsilon)\mathrm{Sopt}(n) + 1$ in $(\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})}$ time, where $\mathrm{Sopt}(n)$ is the optimal solution for the first n items in the input stream, and $App(n)$ is an approximate solution for the first n items in the input stream.*

Proof: Let ϵ be an arbitrary positive constant. Let $\delta = \frac{\epsilon}{1+\epsilon}$. By Lemma 33, we assume that u random elements have been selected from the input elements with size at least $\delta > 0$. We just add all elements with size less than δ into a sum s_1.

If the condition of line 23 in the algorithm Streaming-Bin-Packing is true, then the inequality (93) in Lemma 23 can be satisfied since $h' = \lfloor \frac{n_{\geq\delta}}{m} \rfloor$. Furthermore, as $\theta = 0$, the conditions of Lemma 23 are satisfied. The approximation ratio follows from Lemma 23.

Assume the condition of line 23 is not true in the rest of the proof. Let U be the set of bins for an $(1+\epsilon)$-approximate solution to items of size at least δ by Lemma 16. It takes only $O(m)$ bins to pack those large items since $n_{\geq\varphi}$ is less than v which is $O(m)$. Therefore, we only need $t = O(m)$ bins for packing the items in Y. The final part of the algorithm fills all small items accumulated in s_1 into those bins in U so that each bin has less than δ left. Put all of the items less than δ into some extra bins, and at most one of them has more than δ space left. Filling the small items of size less than δ is to let each bin except the last one waste no more than δ space. This is a fractional way to pack small items. Since the item size is at most δ, and each bin with (fractional) small items has

at least δ space left. The fractional bin packing for adding small items can bring an non-fractional (regular) bin packing. A similar argument is also shown in Lemma 21.

Assume that an optimal solution of a bin packing problem has two types of bins. Each of the first type bin contains at least one item of size δ, and each of the second type bin only contains items of size less than δ. Let V_1 be the set of first type bins, and V_2 be the set of all second type bins. We have that $|U| \leq (1+\epsilon)|V_1|$.

Case 1. If U can contain all items, we have that $|U| \leq (1+\epsilon)|V_1| \leq (1+\epsilon)|V_1 \cup V_2|$.

Case 2. There is a bin beyond those in U is used. Let U' be all bins without more than δ space left. We have that $|U'| \leq \frac{|V_1 \cup V_2|}{(1-\delta)} \leq (1+\epsilon)|V_1 \cup V_2|$. Therefore, the approximate solution is at most $(1+\epsilon)|V_1 \cup V_2| + 1$.

∎

9. Sliding Window Streaming for $S(\delta)$-Bin Packing

A sliding window stream model for bin packing problem is to pack the most recent n items. Select an integer constant λ that is determined by the approximation ratio and δ, the least size of input items. The idea is to start a new session to collect some random items from the input stream after every $\frac{n}{\lambda}$ items.

Assume that $a_{m+1}, \ldots, a_{m+n}$ are the last n input items in the input stream. We maintain a list of sets $S_1, \ldots, S_\lambda$ such that if S_i is a set of random items in $\{a_{m+j_i}, \ldots, a_{m+n}\}$ ($[m+j_i, m+n]$ is called the range of S_i), then the next $S_{(i+1)(mod\ \lambda)}$ is a set of random items in $\{a_{m+j_i+\frac{n}{\lambda}}, \ldots, a_{m+n}\}$. On the other hand, when the range of a set S_i reaches $[m+1, m+n]$), S_i is reset to be empty and starts to collect the random elements from the scratch. We also set a pointer to the set S_i that has the largest range.

After receiving every $\frac{n}{\lambda}$ items in the input stream, the set S_i with the largest range will be passed to the next $S_{i+1(\text{mod }\lambda)}$ if S_i's range reaches size n. The is called rotation, which makes the pointer to the set with the largest range according to the loop $S_1 \to S_2 \to, \ldots, S_{\lambda-1} \to S_\lambda \to S_1$. In the following algorithm we assume that $n = 0 \pmod{\lambda}$. Otherwise, we replace n by $n' = \lceil \frac{n}{\lambda} \rceil \lambda$.

It is easy to see that $n \leq n' \leq n + \lambda$. The bin packing problem for the last n items has a small ratio difference with that for the last n' items if the constant λ is selected large enough.

Algorithm Sliding-Window-Bin-Packing$(c, \eta, k, \gamma, \delta, n)$

Input: bin types constants c, η, and k, a positive constant γ, a streaming of items of size at least δ, and a sliding window size n.

Output: an $(1+\gamma)$-approximation

Steps:

1. Let $\epsilon := \frac{\gamma}{30}$.
2. Let $\theta := 0$.
3. Let μ, ϵ_1 and m are selected by equations (8), (9), and (10), respectively.
4. Let $\lambda := \left\lceil \frac{100}{\gamma\delta} \right\rceil$.
5. Let $t := \frac{n}{\lambda}$.
6. Create t empty sets $S_1, \ldots, S_k$ to hold random elements and make them non-active.
7. Let $u := \frac{c_1 m^2 \log m}{\mu^2}$ be the number of random elements in each S_i according to Lemma 19.
8. Let h_j be the range size of S_j.
9. Start S_1 to be active to collect random elements.

10. Let S_1 hold u copies of the first element a_1 in the stream.

11. For each new element a_i from the input stream ($i = 2, 3, \ldots$)

12. For each active S_j and each of the u items $a_r \in S_j$,

13. replace a_r by a_i with probability $\frac{1}{h_j}$ and let $h_i := h_i + 1$.

14. Let S_j be the set with the largest range h_j.

15. Let $(y_1, \ldots, y_m)$:=Select-Crucial-Items(m, u, S_j) (see Lemma 19).

16. Let $(x_1, \ldots, x_q)$:=Pack-Large-Items(c, η, k, B) with $B = \{y'^h_1, \ldots, y'^h_m\}$ (See Lemma 14).

17. Let y be the cost for the packing with solution $(x_1, \ldots, x_q)$.

18. Output *app* :=Packing-Conversion(n, y) (see Lemma 20).

19. If $i = 0(mod\ t)$

20. then

21. if $i < n$

22. then make $S_{(j+1)\ (\mathrm{mod}\ t)}$ be active, and let $h_{(j+1)(mod\ t)} = 0$.

23. if $i \geq n$

24. then let S_j hold u copies of a_i and let $h_j = 1$ (reset S_j).

End of Algorithm

We have Theorem 36 that shows an example of NP-hard problem that has a constant time and constant space sliding window streaming approximation scheme.

Theorem 35. *Assume that c, η, and k are constants. Let δ be an arbitrary constant. Then Sliding-Window-Bin-Packing(.) is a single pass sliding window streaming randomized approximation algorithm for the $S(\delta)$-bin packing problem with (c, η, k)-related bins that has $O(1)$ updating time and $O(1)$ space, and computes an approximate packing solution App(.) with $\mathrm{Sopt}_{c,\eta,k}(n) \leq App(n) \leq (1+\gamma)\mathrm{Sopt}_{c,\eta,k}(n)$ in $(\frac{1}{\gamma})^{O(\frac{1}{\gamma})}$ time, where $\mathrm{Sopt}_{c,\eta,k}(n)$ is the optimal solution for the last n items in the input stream, and App(n) is an approximate solution for the most recent n items in the input stream.*

Proof: Multiple sessions of groups are generated to maintain the progress of incoming elements. The purpose of the choice of λ at line 4 in Sliding-Window-Bin-Packing(.) is to let it satisfy that $\frac{n}{\lambda} \leq \gamma n\delta/100$ since n items needs at least $m\delta$ bins and $\frac{n}{\lambda}$ items needs at most $\frac{n}{\lambda}$ bins. This control is implemented in lines 19 to 23 in the algorithm Sliding-Window-Bin-Packing(.).

Assume that the n integers in $[1, n]$ represent the last n items from the input stream. Each of the λ groups takes care of the list items in the range $[i \cdot \frac{n}{\lambda} + j, n]$ for $i = 0, \ldots, \lambda - 1$, where j is an integer that moves in the loop $0 \to 1 \to 2 \to 3 \to \ldots \to \frac{n}{\lambda} - 1 \to 0$. We keep λ groups of $u = \frac{c_1 m^2 \log m}{\mu^2}$ random elements each according to Lemma 33, where μ is defined as the proof in Lemma 29 and Lemma 19. After every $\frac{n}{\lambda}$ items, we start picking a new session of elements and drop the oldest session.

When a set S_j holds u random elements from the last h elements for $h \in [n-t, n+t]$, where $t = \frac{n}{\lambda}$. The approximation derived from S_j has a small difference with the optimal solution for the last n elements. Let $\mathrm{Sopt}_{c,\eta,k}(n)$ be the optimal solution for packing the last n items with (c, η, k)-related bins. We have that $\mathrm{Sopt}_{c,\eta,k}(h) - t \leq \mathrm{Sopt}_{c,\eta,k}(n) \leq \mathrm{Sopt}_{c,\eta,k}(h) + t$. By Lemmas 14, 19, and 20, the algorithm outputs an $(1+\gamma/2)$-approximation for $\mathrm{Sopt}_{c,\eta,k}(h)$. By the setting of t, we have that

$(1-\gamma/2)\text{Sopt}_{c,\eta,k}(h) \leq \text{Sopt}_{c,\eta,k}(n) \leq (1+\gamma/2)\text{Sopt}_{c,\eta,k}(h)$. Therefore, an $(1+\gamma/2)$ approximation to $\text{Sopt}_{c,\eta,k}(h)$ is a $(1+\gamma)$ to $\text{Sopt}_{c,\eta,k}(n)$.

∎

Theorem 36. *There is an NP-hard problem that has a constant time and space sliding windows approximation scheme.*

Proof: It follows from Theorem 28 and Theorem 35. ∎

9.1. Constant Time Approximation Scheme for Random Sizes

In this section, we identify more cases of the bin packing problem with constant time approximation. One interesting case is that all items are random numbers in $(0, 1]$.

Definition 37. Let δ_1, δ_2 and ϵ_1 are positive parameters. For a list $a_1, \ldots, a_n$ of input of bin packing problem, it has the $(\delta_1, \delta_2, \epsilon_1)$-*property* if the list $a_1, \ldots, a_n$ satisfies

$$\left\lceil \frac{\delta_2}{c-\delta_2} |\{i : a_i \leq \delta_2 \text{ and } a_i \in \{a_1, \ldots, a_n\}\}| \right\rceil \leq \epsilon_1 \eta \delta_1 |\{i : a_i \geq \delta_1 \text{ and } a_i \in \{a_1, \ldots, a_n\}\}|.$$

Theorem 38. *Let δ_1, δ_2 and ϵ are positive constants with $\delta_2 \geq \delta_1$. Then there is a constant $(\frac{1}{\epsilon\delta_1})^{O(\frac{1}{\delta_1})}$ time algorithm such that if the bin packing problem with (c, η, k)-related bins and $(\delta_1, \delta_2, \epsilon/3)$-property, it gives an $(1+\epsilon)$-approximation.*

Proof: Let t_1 be the cost of an optimal solution to pack those items of size at least δ_1 and t_2 be the cost of an optimal solution to pack those items of size at most δ_2. Let t be the cost of an optimal solution to pack all items in the list. Clearly, we have $t \geq t_1$.

The number of bins is at least $b_1 = \delta_1 |\{i : a_i \geq \delta_1 \text{ and } a_i \in \{a_1, \ldots, a_n\}\}|$ for packing those items of size at least δ_1. The cost for packing those items of size at least δ_1 is at least ηb_1 since the least cost is η among all bins. Thus, $\eta b_1 \leq t_1$. The number of bins for packing those items of size at most δ_2 is at most $b_2 = \left\lceil \frac{\delta_2}{c-\delta_2} |\{i : a_i \leq \delta_2 \text{ and } a_i \in \{a_1, \ldots, a_n\}\}| \right\rceil$ since at most one bin wastes space more than δ_2. The cost for packing those items of size δ_2 is at most b_2 since 1 is the upper bound of the largest cost bin.

With $(\frac{1}{\epsilon\delta_1})^{O(\frac{1}{\delta_1})}$ time, we derive an $(1+\frac{\epsilon}{3})$-approximation b_1' for the items of size at least δ_1 by Theorem 30. We have $b_1 \leq b_1'$ since b_1' is an approximation to the optimal solution and b_1 is a lower bound of the optimal solution for packing items of size at least δ_1. The cost for the bins for packing those items of size at most δ_2 is at most $b_2 \leq \frac{\epsilon}{3}\eta b_1 \leq \frac{\epsilon}{3}\eta b_1'$ because of the $(\delta_1, \delta_2, \epsilon/3)$-property. We output the approximation with cost $b_1' + \frac{\epsilon}{3}\eta b_1'$. We have

$$\begin{aligned} b_1' + \frac{\epsilon}{3}\eta b_1' &\leq (1+\frac{\epsilon}{3})t_1 + \frac{\epsilon}{3}(1+\frac{\epsilon}{3})t_1 \qquad (\text{note } \eta \leq 1) \\ &\leq (1+\epsilon)t_1 \\ &\leq (1+\epsilon)t. \end{aligned}$$

Therefore, we derive an $(1+\epsilon)$-approximation for packing the input list with (c, η, k)-related bins. ∎

Theorem 39. *Assume that c, η, and k are constants. Assume that a and b with $a < b \leq c$ are two constants in $[0, 1]$. Let ϵ be a constant in $(0, 1]$. Then there is a randomized constant $(\frac{1}{\epsilon})^{O(\frac{1}{(a+\epsilon)})}$ time approximation scheme for the bin packing problem with (c, η, k)-related bins that each element is a random element from $[a, b]$.*

Proof: Let ϵ_2 be a constant in $(0, \frac{1}{4})$ and will be determined later. Let $\epsilon_1 = \frac{\epsilon}{3}$. Let $\delta_1 = \delta_2 = a + \epsilon_2(b-a)$. We prove that a list with random elements from $[a,b]$ satisfies $(\delta_1, \delta_2, \epsilon_1)$-property for all large n with high probability. Assume that $a_1, \ldots, a_n$ is a list of random elements in $[a,b]$.

We note that with probability 0, a random element a_i from $[a,b]$ is equal to a. For each random element $a_i \in [a,b]$, with probability $p_1 = 1 - \epsilon_2$, we have $a_i \geq \delta_1$. By Theorem 4, with probability at most $P_1 = g_1(\frac{1}{4})^{p_1 n}$, $n_1 = |\{i : a_i \geq \delta_1\}|$ is less than $(p_1 - \frac{1}{4})n$ elements. We note $(p_1 - \frac{1}{4})n \geq \frac{n}{4}$ since $p_1 \geq \frac{1}{2}$.

For each random element $a_i \in [a,b]$, with probability $p_2 = \epsilon_2$, we have $a_i < \delta_2$. By Theorem 5, with probability at most $P_2 = g_2(1)^{p_2 n}$, we have $n_2 = |\{i : a_i < \delta_2\}|$ is more than $(1+1)p_2 n = 2\epsilon_2 n$.

Assume that $n_1 \geq \frac{n}{4}$ and $n_2 \leq 2\epsilon_2 n$.

Since ϵ_2 is a constant in $(0, \frac{1}{4})$, we have $\delta_2 \leq a + \frac{1}{4}(b-a)$. Thus, we have

$$\begin{aligned}
\frac{\delta_2}{c - \delta_2} &\leq \frac{\delta_2}{b - \delta_2} \\
&\leq \frac{b}{b - \delta_2} \\
&\leq \frac{b}{b - (a + \frac{1}{4}(b-a))} \\
&\leq \frac{4b}{3(b-a)}.
\end{aligned}$$

Assume that n is large enough such that $(\frac{1}{b-a})\epsilon_2 n \geq 1$. We have that

$$\begin{aligned}
\left\lceil \frac{\delta_2}{c - \delta_2} n_2 \right\rceil &\leq \frac{\delta_2}{c - \delta_2} n_2 + 1 \\
&\leq \frac{4b}{3(b-a)} n_2 + 1 \\
&\leq \frac{4b}{3(b-a)} \cdot 2\epsilon_2 n + 1 \\
&\leq \frac{16b}{3(b-a)} \epsilon_2 n \\
&\leq \epsilon_1 \eta \delta_1 \frac{n}{4} \\
&\leq \epsilon_1 \eta \delta_1 n_1,
\end{aligned}$$

where ϵ_2 is selected to be $\frac{3\epsilon_1 \eta \delta_1 (b-a)}{64b}$, which is less than $\frac{1}{4}$. Therefore, with probability at most $P_1 + P_2$, the $(\delta_1, \delta_2, \epsilon_1)$-property is not satisfied. Theorem 39 follows from Theorem 38. ∎

Theorem 40. *Assume that $a < b$ are two constants in $[0,1]$. Then there is a randomized constant $(\frac{1}{\epsilon})^{O(\frac{1}{a+\epsilon})}$ time approximate scheme for the bin packing problem that each element is a random element from $[a,b]$.*

Proof: It follows from Theorem 39. ∎

10. Conclusions

This paper shows a dense hierarchy of approximation schemes for the bin packing problem which has a long history of research. Pursing sublinear time algorithm brings a better understanding about the technology of randomization, and also gives some new insights about the problems that may already have linear time solution. Our sublinear time algorithms are based on an adaptive random sampling method for the bin packing problem developed in this paper. The hierarchy approach, which is often used in the complexity theory, may give a new way for algorithm analysis as it gives more information than the worst case analysis from the classification.

11. Acknowledgements

We would like to thank Xin Han for his helpful suggestions which improves the presentation of this paper. This research is supported in part by National Science Foundation Early Career Award 0845376. An earlier version of this paper is posted at http://arxiv.org/abs/1007.1260.

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the symposium on theory of computing*, pages 20–29, 1996.

[2] D. Applegate, L. Buriol, B. Dillard, D. Johnson, and P. Shore. The cutting-stock approach to bin packing: Theory and experiments. In *Proceedings of Algorithm Engineering and Experimentation (ALENEX)*, pages 1–15, 2003.

[3] T. Batu, P. Berenbrink, and C. Sohler. A sublinear-time approximation scheme for bin packing. *Theoretical Computer Science*, 410:5082–5092, 2009.

[4] D. Brown. A lower bound for on-line one-dimensional bin packing problem. Technical Report 864, University of Illinois, Urbana, IL, 1979.

[5] B. Chazelle, D. Liu, and A. Magen. Sublinear geometric algorithms. *SIAM Journal on Computing*, 35:627–646, 2005.

[6] B. Chazelle, R. Rubfinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on computing*, 34:1370–1379, 2005.

[7] J. Csirik, D. Johnson, C. Kenyon, J. Orlin, P. Shore, and R. Weber. A self-organizing bin packing heuristic. In *Proceedings of algoirthm engineering and experimentation (ALENEX)*, pages 246–265, 1999.

[8] J. Csirik, D. Johnson, C. Kenyon, J. Orlin, P. Shore, and R. Weber. On the sum-of-squares algorithm for bin-packing. In *Proceedings of the 22nd annual ACM symposium on theory of computing (STOC)*, pages 208–217, 2000.

[9] A. Czumaj, F. Ergun, L. Fortnow, I. N. A. Magen, R. Rubinfeld, and C. Sohler. Sublinear approximation of euclidean minimum spanning tree. *SIAM Journal on Computing*, 35:91109, 2005.

[10] A. Czumaj and C. Sohler. Estimating the weight of metric minimum spanning trees in sublinear-time. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 175–183, 2004.

[11] W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within 1+epsilon in linear time. *Combinatorica*, 1(4):349–355, 1981.

[12] P. Flajolet and G. Martin. Probabilistic counting algorithms for data base application. *Journal of computer and system sciences*, 31:182–209, 1985.

[13] B. Fu and Z. Chen. Sublinear-time algorithms for width-bounded geometric separators and their applications to protein side-chain packing problems. *Journal of Combinatorial Optimization*, 15:387–407, 2008.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Company, New York, 1979.

[15] M. Gilmore and R. Gomory. A linear programming approach to the cutting-stock problem - part ii. *Operations Research*.

[16] M. Gilmore and D. Johnson. A linear programming approach to the cutting-stock problem. *Operations Research*.

[17] O. Goldreich and D. Ron. On testing expansion in bounded-degree graphs. Technical Report 00-20, Electronic Colloquium on Computational Complexity, http://www.eccc.uni-trier.de/eccc/, 2000.

[18] M. Li, B. Ma, and L. Wang. On the closest string and substring problems. *Journal of the ACM*, 49(2):157–171, 2002.

[19] F. Liang. A lower bound for on-line bin packing. *Information processing letters*, 10:76–79, 1980.

[20] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 2000.

[21] J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theoretical computer science*, 12:315–323, 1980.

[22] S. G. O. Goldreich and D. Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45:653750, 1998.